Univerzitet u Beogradu

Matematički fakultet

Master rad

Korišćenje Selenium-a za testiranje softvera vezanog za korisnički interfejs finansijske aplikacije

Beograd, septembar 2016.

Kandidat: Suzana Radotić

Tema: Korišćenje Selenium-a za testiranje softvera vezanog za korisnički interfejs finansijske aplikacije

Mentor: dr Dušan Tošić

Članovi komisije: dr Miroslav Marić

dr Vladimir Filipović

Apstrakt:

Selenium je skup alata za automatizovano testiranje korisnickog interfejsa Web aplikacija. Omogućava testiranje Web aplikacija u gotovo svim dostupnim pretraživačima, dok test skriptovi mogu biti pisani u različitim programskim jezicima. Selenium se lako integriše sa drugim alatima i omogućava istovremeno izvršavanje test skriptova na više različitih platformi.

U ovom radu opisano je automatizovano testiranje korisničkog interfejsa finansijske aplikacije korišćenjem Selenium IDE i Selenium WebDriver komponenti Selenium-a. Napisani su i implementirani automatski testovi za testiranje ponašanja aplikacije koja se koristi za dodavanje i izmenu podataka o klijentima i njihovim računima, kao i naprednu pretragu njihovih transakcija.

Abstract:

Selenium is a set of tools for UI automated testing of Web applications. It can be used for Web application testing in almost all currently available Web browsers, while test scripts themselves can be coded in different programming languages. Selenium is easy to integrate with other tools, and provides simultaneous test script execution on many different platforms.

This Thesis covers automated UI testing of a financial application, using Selenium IDE and Selenium WebDriver components of Selenium. It describes the implementation of tests for adding and modifying client and account data, as well as for the advanced transaction search functionality.

Sadržaj

1 Uvod

Svedoci smo da poslednjih nekoliko godina postajemo sve više tehnički orjentisano društvo i da gotovo nema aspekta u našim životima u kome ne koristimo neku vrstu softvera. Upravo iz tog razloga, softverski sistemi postaju sve važniji, kompleksniji i raste značaj njihovog kvaliteta. Osiguranje kvaliteta softvera važno je koliko i njegov razvoj jer greške koje se javljaju u softveru ponekad mogu da dovedu do ozbiljih problema, na primer – kod softvera koji se koristi za kontrolu leta, softvera koji se primenjuje u medicinske svrhe ili finansijskog softvera.

Testiranje softvera predstavlja samo jedan deo procesa kontrole kvaliteta softvera. Različite metode koriste se za testiranje različitih slojeva softvera, a ono što je svima zajedničko jeste da se mogu izvoditi manuelno ili automatski korišćenjem posebnih alata. I jedan i drugi način imaju svoje prednosti i mane, ali se u poslednje vreme sve više teži automatizaciji jer se automatizacijom testiranja skraćuje vreme potrebno za testiranje.

Selenium je alat za automatizovano testiranje korisničnog interfejsa Web aplikacija. Zahvaljujući svojoj univerzalnosti i tome što podržava testiranje u gotovo svim dostupnim pretraživačima, Selenium je trenutno najčešći izbor stručnjaka koji se bave testiranjem korisničkog interfejsa Web aplikacija. U ovom radu opisano je korišćenje Selenium-a za automatizovano testiranje korisničnog interfejsa jedne finansijske aplikacije. Međutim, sličan postupak se koristi i u drugim aplikacijama.

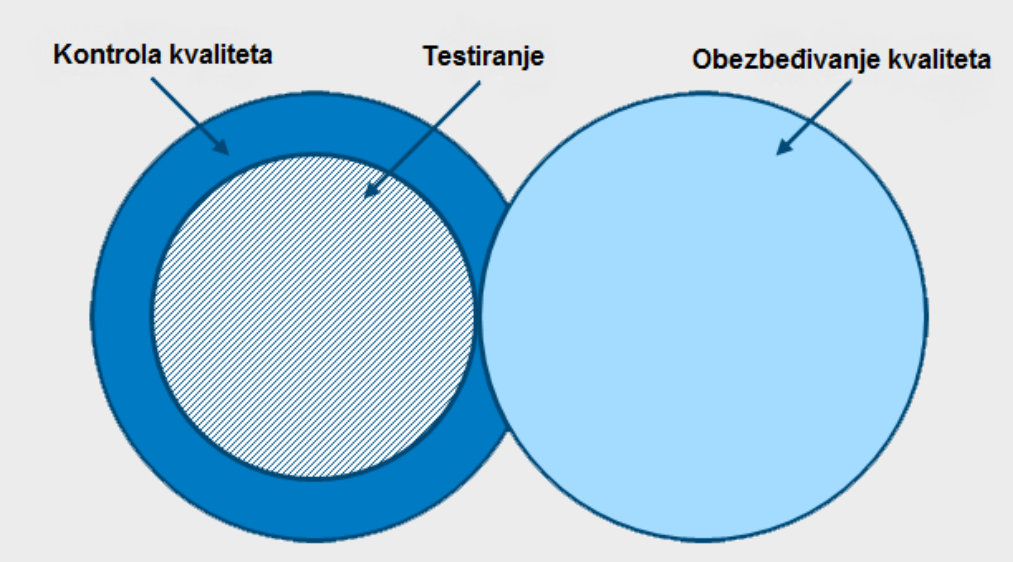
2 O testiranju softvera

2.1 Obezbeđivanje kvaliteta softvera, kontrola kvaliteta i testiranje softvera

Pojmovi obezbeđivanja kvaliteta softvera (eng. Software Quality Assurance), kontrole kvaliteta (eng. Quality Control) i testiranja softvera (eng. Software Testing) se često koriste u istom kontekstu. Međutim, postoji jasna razlika među ovim pojmovima ([1]):

- **Obezbeđivanje kvaliteta softvera** obuhvata skup aktivnosti potrebnih da bi se obezbedilo adekvatno poverenje da će proizvod procesa razvoja softvera ili unapređenja softvera zadovoljiti početne tehničke zahteve i, da će, u isto vreme, dogovorena novčana sredstva i vreme potrebno za razvoj biti ispoštovani.
  Proces obezbeđivanja kvaliteta softvera počinje pre samog razvoja softvera i predstavlja garanciju da će se čitav proces razvoja odvijati u skladu sa zahtevima klijenta, planovima i usvojenim procedurama. Cilj obezbeđivanja kvaliteta softvera jeste da se preduprede greške.
- **Kontrola kvaliteta** je skup aktivnosti koje se sprovode na kraju procesa razvoja softvera i kontrolom se upoređuje kvalitet napravljenog softvera sa polaznim zahtevima klijenta.
  Kontrola kvaliteta softvera počinje tek kada je proizvod napravljen (ili, u slučaju statičkog testiranja, o čemu će tek biti reči, kada je dokument napisan). Cilj ovog procesa jeste nalaženje softverskih grešaka i obezbeđivanje njihovog ispravljanja.
- **Testiranje** je deo kontrole kvaliteta, ali postoje aktivnosti kontrole kvaliteta koje ne predstavljaju testiranje kao što su provera koda, tehnička revizija i sl.

Odnos između ovih pojmova prikazan je vizuelno na slici 1.



Slika 1: Odnos između obezbeđivanja kvaliteta, kontrole kvaliteta i testiranja

2.2 Proces testiranja softvera

Testiranje softvera je proces koji je usmeren ka procenjivanju karakteristika i sposobnosti softvera ili sistema i utvrđivanju da li taj softver ili sistem ispunjava određene uslove. Ovaj proces uključuje izvršavanje programa ili aplikacije sa namerom pronalaženja softverskih grešaka (eng. bug), i svakog vida odstupanja sistema od originalnih zahteva korisnika, u toku izvršavanja programa.

Najkraće rečeno, testiranje je proces određivanja nivoa kvaliteta softvera. Nivo kvaliteta softvera i zahtevi variraju od sistema do sistema, ali neke osnovne karakteristike koje svi oni treba da poseduju i koje se vrednuju su: pouzdanost, stabilnost, prenosivost, kompatibilnost i upotrebljivost.

Testiranje je usko povezano sa pojmovima verifikacije i validacije softvera ([2]).

Verifikacija je proces procene softvera ili njegovih komponenti sa ciljem da se utvrdi da li proizvodi određenih faza razvoja zadovljavaju uslove sa početka tih faza. Ovim procesom se utvrđuje koliko je proizvod jedne faze kompatibilan sa proizvodom prethodnih faza i da li svi delovi sistema, pojedinačno i kao celina, rade ispravno. Dakle, pitanje na koje se verifikacijom daje odgovor je: „Da li se softver razvija ispravno?”. Verifikacija se vrši revizijom planova, zahteva, specifikacija i upoređivanjem realizovanog koda sa zahtevima korisnika.

Validacija se definiše kao proces procene softvera ili njegovih komponenti, tokom ili na kraju njegovog razvoja, sa ciljem da se utvrdi da li zadovljava polazne zahteve. Cilj

validacije je da odgovori na pitanje da li se razvija pravi softver, to jest, softver koji je u skladu sa zahtevima korisnika definisanim na početku procesa razvoja i da li je takav softver upotrebljiv za korisnika. Validaciju obavljaju testeri kroz izvršavanje test skriptova.

Verifikacija se označava kao statičko testiranje softvera, dok validacija predstavlja dinamičko testiranje softvera. Dakle, proces testiranja se može posmatrati kao kombinacija statičkog i dinamičkog pristupa ([5]).

Statičko testiranje je testiranje bez izvršavanja koda. Sprovodi se u različitim fazama razvoja softvera, najviše u ranim, sa ciljem poboljšanja kvaliteta proizvoda koji se testira i što ažurnijeg otklanjanja grešaka. Obično se vrši kroz razne vrste pregledanja i revizija čiji predmet može biti dokumentacija ili kôd. Druga vrsta statičkog testiranja je statička analiza koda, kao što je provera sintaksne ispravnosti koda ili analiza složenosti koda. Statičko testiranje omogućuje detektovanje grešaka u ranim fazama razvoja softvera.

Za razliku od statičkog, dinamičko testiranje predstavlja proces provere, koji se sprovodi u toku izvršavanja programa. Dinamičkim testiranjem se vrši provera ponašanja sistema, tačnije upoređivanje dobijenog i očekivanog izlaza. Dinamičko testiranje softvera podrazumeva i testiranje raznih kvalitativnih aspekata softvera kao što su testiranje performansi sistema, testiranje pouzdanosti sistema, testiranje sistema pod velikim opterećenjima, poznatija kao stres testiranja i mnoga druga.

Za početak procesa testiranja, postojanje koda nije neophodno. Dovoljno je imati samo jasno definisane zahteve korisnika jer testiranje počinje analizom tih zahteva. Nakon toga sledi planiranje čitavog procesa testiranja, kreiranje test skriptova, priprema za izvršavanje skriptova, njihovo izvršavanje i, na kraju, procena statusa uz završetak testiranja. Dakle, sve aktivnosti koje testiranje obuhvata se mogu grupisati u nekoliko faza ([4]):

1) Planiranje testiranja – Tokom ove faze određuju se opseg testiranja, pristup, strategije i metode testiranja, kao i kriterijum završetka testiranja. Takođe se odlučuje i o potrebnim resursima i dogovara način komunikacije između članova tima.

2) Analiza i dizajn testova – Onog momenta kada je potpuno jasno šta i kako treba da se uradi, može se pristupiti detaljnoj analizi zahteva korisnika. Ispituje se mogućnost testiranja određenih delova koda, prikupljaju potrebni podaci i preciziraju zahtevi korisnika. Kada je očekivano ponašanje sistema jasno, pristupa se kreiranju test scenarija i test skriptova.

3) Implementacija i izvršavanje testova – Ova faza podrazumeva određivanje prioriteta izvršavanja testova, pripremu testova za automatizovano testiranje (ukoliko je ono deo procesa testiranja) i organizaciju testova za što efikasnije izvršavanje. Nakon toga sledi izvršavanje testova pod različitim uslovima, čuvanje rezultata testiranja, logovanje nađenih grešaka u ponašanju sistema, praćenje njihovog životnog ciklusa, retestiranje i ponovno izvršavanje testova posle popravke grešaka.

4) Procena kriterijuma završetka testiranja i izveštavanje – Testiranje se može smatrati beskonačnim procesom, jer svaka izmena u kodu, čak i koja podrazumeva popravljanje grešaka, može da dovede do novih grešaka. Iz tog razloga se, za različite oblasti testiranja, definiše kriterijum završetka testiranja u odnosu na rezultate izvršavanja test skritpova, procenta nerešenih bagova ili preostalog vremena za testiranje. Po završetku testiranja piše se izveštaj koji se koristi kao dokaz da je testiranje bilo uspešno.

5) Aktivnosti zatvaranja testiranja – Testiranje se zatvara kada je softver isporučen korisniku, mada može se desiti i u nekim drugim situacijama, na primer, kada je projekat otkazan ili je neki cilj postignut. Tokom ove faze, test skriptovi i dokumentacija se arhiviraju, dok se primenjeni proces testiranja analizira i diskutuje o tome šta je bilo dobro, a šta ne.

2.3 Metode testiranja softvera

U odnosu na način na koji se posmatra sistem koji se testira, postoje dva pristupa testiranju – tehnika bele kutije (eng. White-Box Testing, Glass-Box Testing ili Clear-Box Testing) i tehnika crne kutije (Black-Box Testing) ([3]).

U slučaju tehnike bele kutije, unutrašnja struktura sistema za testiranje je poznata. Tester ima pristup kodu softvera koji se testira i razume način njegove implementacije. Fokus je na strukturnim elementima kao što su naredbe i petlje. Cilj testiranja jeste da se ispita da li postoje greške u strukturi programa. S obzirom na to da kreiranje test skriptova, njihovo izvršavanje i analiza rezultata zahtevaju mnogo vremena, ovakav način testiranja se obično primenjuje na delove koda, a retko na celokupan kôd. Korisno je za nalaženje grešaka u dizajnu, logičkih grešaka i grešaka u toku podataka.

Kod tehnike crne kutije ne koriste se znanja o internoj strukturi softvera, već je fokus na funkcionalnim zahtevima. Tester ceo sistem posmatra kao crnu kutiju i zna samo šta softver radi, ali ne i kako radi. Različiti ulazni podaci se koriste za izvršavanje testova, a onda se rezultat upoređuje sa očekivanim rezultatom navedenim u specifikaciji.

Pored navedenih, postoji i tehnika sive kutije (eng. Gray-Box Testing) u kojoj se koristi uvid u unutrašnju strukturu softvera prilikom kreiranja test skriptova, dok se izvršavanje tih skriptova odvija tehnikom crne kutije. Ovaj metod je koristan u slučajevima testiranja integracije različitih delova koda.

2.4 Nivoi testiranja softvera

Testiranje softvera, posebno velikih sistema, vrši se na više nivoa, od kojih svaki ima specifične ciljeve testiranja. Najčešći nivoi su ([1] i [6]):

- jedinično testiranje
- integraciono testiranje
- sistemsko testiranje
- testiranje prihvatljivosti

Jedinično testiranje (eng. Unit Testing) je testiranje funkcionalnosti programskih komponeti (jedinica) nezavisno od ostalih delova sistema. Jedinicama programa se smatraju funkcije ili klase. Programer koji piše kôd obično i izvodi jedinično testiranje svog dela koda u razvojnom okruženju koristeći neki od okvira za jedinično testiranje. Cilj ovakvog načina testiranja jeste da se detektuju funkcionalne i strukturne greške u odgovarajućoj jedinici, koje se odmah po nalaženju i poprave.

Integraciono testiranje (eng. Integration Testing) je testiranje interakcije između različitih jedinica sistema ili interakcije između čitavih sistema. Obično se obavlja po završetku testiranja svake od komponenti koje se integrišu. Cilj integracionog testiranja je da proveri da li sistemske komponente sarađuju kao što je to predviđeno u definisanim specifikacijama sistema.

Sistemsko testiranje (eng. System Testing) je testiranje sistema kao celine od strane test tima i u posebnom test okruženju. Često predstavlja poslednju fazu testiranja pre nego što se softver isporuči klijentu. Test okruženje treba da odgovara produkcionom okruženju, koliko god je to moguće, da bi se smanjio rizik javljanja grešaka koje se teško detektuju zbog specifičnosti test okruženja. Cilj sistemskog testiranja je procena funkcionalnog ponašanja sistema i nefunkcionalnih osobina. U nefunkcionalne spadaju performanse i pouzdanost sistema, dok se funkcionalni zahtevi uglavnom testiraju metodom crne kutije.

Testiranje prihvatljivosti (eng. Acceptance Testing) je faza koja se obavlja kada se softver pravi za određenu namenu i određenog klijenta. Kod ovog nivoa testiranja sam klijent daje sud o tome da li su ispunjeni svi zahtevi i da li isporučeni softver obavlja traženu svrhu. Često uključuje alfa i beta testiranje. Alfa testiranje se vrši tokom samog razvoja programa kad se programerima odmah ukazuje na greške. Beta testiranje se vrši onda kada su sve moguće softverske greške nađene i popravljene i obavlja se u realnom okruženju u kome softver i treba da se koristi.

2.5 Tipovi testiranja softvera

U zavisnosti od toga šta je predmet testiranja, testiranje može biti funkcionalno, nefunkcionalno i testiranje posle izmena koda ([1]).

Funkcionalno testiranje je provera da li se sistem ili neka njegova komponenta ponašaju onako kako je to definisano zahtevima. Za različite ulazne parametre treba utvrditi da li se dobijeni rezultat razlikuje od očekivanog. Pri tom se, takođe, uzimaju u obzir i parametri koji ne treba da budu podržani na osnovu zahteva, jer sistem mora da ima rešenje i u tom slučaju (takozvani negativan test).

Nefunkcionalno testiranje podrazumeva testiranje nefunkcionalnih karakteristika sistema. To su osobine koje se mogu izmeriti, na primer, koliko dugo treba da čekamo da bismo videli rezultat neke akcije ili koliko ljudi može biti ulogovano na sistem u isto vreme. U nefunkcionalno testiranje ubrajaju se testiranje performansi, testiranje opterećenja, testiranje sigurnosti, testiranje pouzdanosti, testiranje prenosivosti i slično.

Testiranje posle izmena koda – Prilikom svake izmene koda, stanje sistema se menja. Zbog toga je, pre bilo kakve detaljnije provere softvera, potrebno testirati osnovne funkcionalnosti (eng. Smoke Testing). Ukoliko nema kritičnih softverskih grešaka, može se nastaviti sa testiranjem, a ukoliko neka od osnovnih funkcionalnosti nije ispunjena, ona obično blokira testiranje i mora se popraviti u najkraćem roku.

Regresiono testiranje je još jedan vid testiranja koji se obavlja posle velikih izmena koda i cilj ovog testiranja jeste da se utvrdi da se deo koda, koji ne treba da se menja, zaista i nije promenio, to jest, da nema neočekivanih grešaka. Ova vrsta testiranja obavezno se izvodi na kraju svakog ciklusa testiranja pre nego što se kôd pošalje u produkciono okruženje. Kako svaka izmena koda, uključujući i popravku grešaka, nosi sa sobom rizik pojave novih grešaka, regresiono testiranje treba ponavljati iznova i iznova, što oduzima mnogo vremena. Zbog toga je najbolje da se ova vrsta testiranja automatizuje.

2.6 Manuelno i automatizovano testiranje softvera

Manuelno testiranje podrazumeva ručno izvršavanje test skritpova različitim alatima u kojima se prate koraci testa i beleže test rezultati. Izvršavanje testa se smatra uspešnim ako je ponašanje sistema koji se testira u skladu sa očekivanim ponašanjem.

Nasuprot manuelnom je automatizovano testiranje koje zahteva postojanje određenog koda koji je napisan da bi se automatizovali koraci pri izvršavanju određenog test skirpta.

Manuelno i automatizovano testiranje prate iste faze procesa testiranja i mogu se primeniti na različitim novima i tipovima testiranja. Oba pristupa imaju svoje prednosti i mane, a neke od njih su:

1) Automatizovano testiranje je brže, što dovodi do velike uštede vremena. To je posebno značajno kod izvršavanja regresionih testova kojih je obično veliki broj i koji se moraju često ponavljati. Sem toga, izvršavanje automatskih testova ne zahteva ljudsko prisustvo pa je moguće njihovo 24-časovno izvršavanje.

2) Automatizovano testiranje je pouzdanije od manuelnog testiranja zato što su kod manuelnog testiranja mogući propusti usled umora ili pada koncentracije testera. Sem toga, manuelno testiranje se uvek oslanja na mišljenje osobe koja testira. Takođe, ograničeni resursi u procesu testiranja često onemogućavaju efikasno i blagovremeno manuelno testiranje.

3) Automatizovano testiranje je skuplje od manuelnog testiranja i zahteva dodatno obučeno osoblje. Ukoliko se implementacija često menja, održavanje testova može biti jako naporno.

4) Postoje situacije u kojima je manuelno testiranje bolji izbor od automatizovanog - na primer, ukoliko se korisnički interfejs menja često, manuelno testiranje je pogodnije iz razloga što svaka promena korisničkog interfejsa zahteva pisanje novih automatskih testova. Zatim, ukoliko nema dovoljno vremena za pisanje automatskih testova, manuelno testiranje je efikasnije ili ako je upotreba neke Web aplikacije kratkotrajna, a ne postoji nijedan adekvatno napisan test za testiranje iste, manuelno testiranje se smatra efikasnijim rešenjem.

3 Selenium

Selenium je skup alata različitih mogućnosti za automatizovano testiranje Web aplikacija. Zahvaljući raznovrsnosti alata, od kojih svaki ima drugačiji pristup testiranju i rešavanju specifičnih problema automatizacije, Selenium podržava testiranje Web aplikacija u gotovo svim dostupnim pretraživačima. Test skriptovi mogu biti pisani u različitim programskim jezicima kao što su C#, Java, Ruby, Python i Perl i pokretani na Windows, Macintosh ili Linux platformama. Selenium je softver otvorenog koda, koji se potpuno besplatno može koristiti ([7]).

3.1 Razvoj Selenium-a

Prva verzija Selenium-a nastala je 2004. godine kada je Džejson Hagins (Jason Huggins), za potrebe testiranja aplikacije u firmi ThoughtWorks, u kojoj je u to vreme radio, napisao

Javascript biblioteku koju je integrisao sa Web stranama, što mu je omogućilo automatsko izvršavanje testova u različitim pretraživačima. Ta biblioteka je osnova na kojoj počiva Selenium Remote Control (poznat i kao Selenium 1), na kome je Hagins, zajedno sa drugim inženjerima, nastavio da radi. Selenium Remote Control predstavlja prvi alat koji, korišćenjem programskog jezika po izboru, može da kontroliše pretraživač ([7] i [8]).

Međutim, Selenium zasnovan na Javascript-u imao je svoje nedostatke. Zbog bezbednosnih mera pretraživača prema Javascript-u i sve složenijih aplikacija koje su uključivale nova ograničenja, mnoge stvari je bilo nemoguće uraditi.

Sajmon Stjuart (Simon Stewart), inženjer u Guglu, počeo je 2006. godine da radi na projektu koji je nazvao WebDriver. Testeri u Guglu su intenzivno koristili Selenium i morali su da pronalaze različite načine da prevaziđu njegove nedostatke. Otuda Stjuartova ideja da napravi alat koji će direktno komunicirati sa pretraživačem, zaobilazeći Javascript.

S obzirom da WebDriver pruža moćnije mogućnosti za testiranje od Selenium-a, a Selenium, sa druge strane, podržava veći broj pretraživača, tvorci Selenium-a 1 i WebDrivera-a, odlučili su 2009. godine da ova dva projekta spoje u jedan – Selenium WebDriver ili Selenium 2, ponudivši korisnicima najbolje moguće rešenje.

Paralelno, 2008. godine, Filip Hanrigu (Philippe Hanrigou), koji je takođe radio u ThoughtWorks-u, napravio je Selenium Grid koji podržava istovremeno izvršavanje više Selenium testova na bilo kom broju lokalnih ili udaljenih sistema, značajno umanjujući, na taj način, vreme potrebno za izvršavanje testova.

Naziv „Selenium” je nastao iz šale koju je napravio Džejson Hagins na račun konkurentske firme Merkjuri (Mercury). Merkjuri je u to vreme plasirao Mercury Interactive QuickTest Professional alat za testiranje. Hagins je želeo da ponudi alternativu i za svoj alat želeo ime koje to i pokazuje. Kako „mercury” u prevodu znači živa, novi alat dobio je naziv „Selenium”, jer se trovanje živom leči selenom (na engleskom „selenium”).

3.2 Komponente Selenium-a

Komponente koje Selenium uključuje su:

- Selenium IDE
- Selenium WebDriver
- Selenium Grid

Sledi opis Selenium IDE i Selenium WebDriver alata, koji će biti korišćeni za automatizaciju testova ([8]).

3.2.1 Selenium IDE

Selenium IDE (eng. Integrated Development Environment) je alat za kreiranje test skriptova koji se automatski izvršavaju. To je, zapravo, Mozilla Firefox dodatak (eng. plugin) koji omogućava snimanje, modifikaciju i debagovanje test skriptova. Vrlo je efikasan jer pruža mogućnost snimanja akcija onim redosledom kojim treba da se testiraju. Dobar je za početak učenja automatizovanog testiranja jer je od sva tri Selenium alata najjednostavniji za upotrebu i upoznavanje komandi.

Skriptovi mogu da se snimaju automatski, ali se mogu i ručno pisati, budući da je svaki test skipt u Selenium-u niz komandi. Snimanjem se dobija kôd u jeziku koje se zove Seleniz (eng. Selenese), posebnom Selenium jeziku za pisanje test skriptova. Skup komandi Seleniza omogućava testiranje Web aplikacija sa različitih aspekata, na primer, testiranje prisustva elemenata korisničkog interfejsa na osnovu HTML tagova, testiranje opcija u menijima, testiranje polja za unos teksta, popunjavanje i slanje formi, podataka u tabelama, ali i testiranje veličine prozora ili akcija kao što su klik na link ili klik mišem.

Snimljeni test skriptovi se podrazumevano čuvaju u HTML formatu. Međutim, u zavisnosti od potreba korisnika, skriptovi mogu da se eksportuju u format nekog od podržanih programskih jezika – C#, Java, Python, Ruby, pri čemu se, obično, nadograđuju funkcijama izabranog programskog jezika.

Selenium IDE se koristi za snimanje i izvršavanje test skriptova isključivo u Mozilla Firefox pretraživaču, ali snimljeni test skriptovi mogu da se izvršavaju i u drugim pretraživačima korišćenjem Selenium WebDriver-a.

Instalacija Selenium IDE-a

Selenium IDE se instalira na jednostavan način:

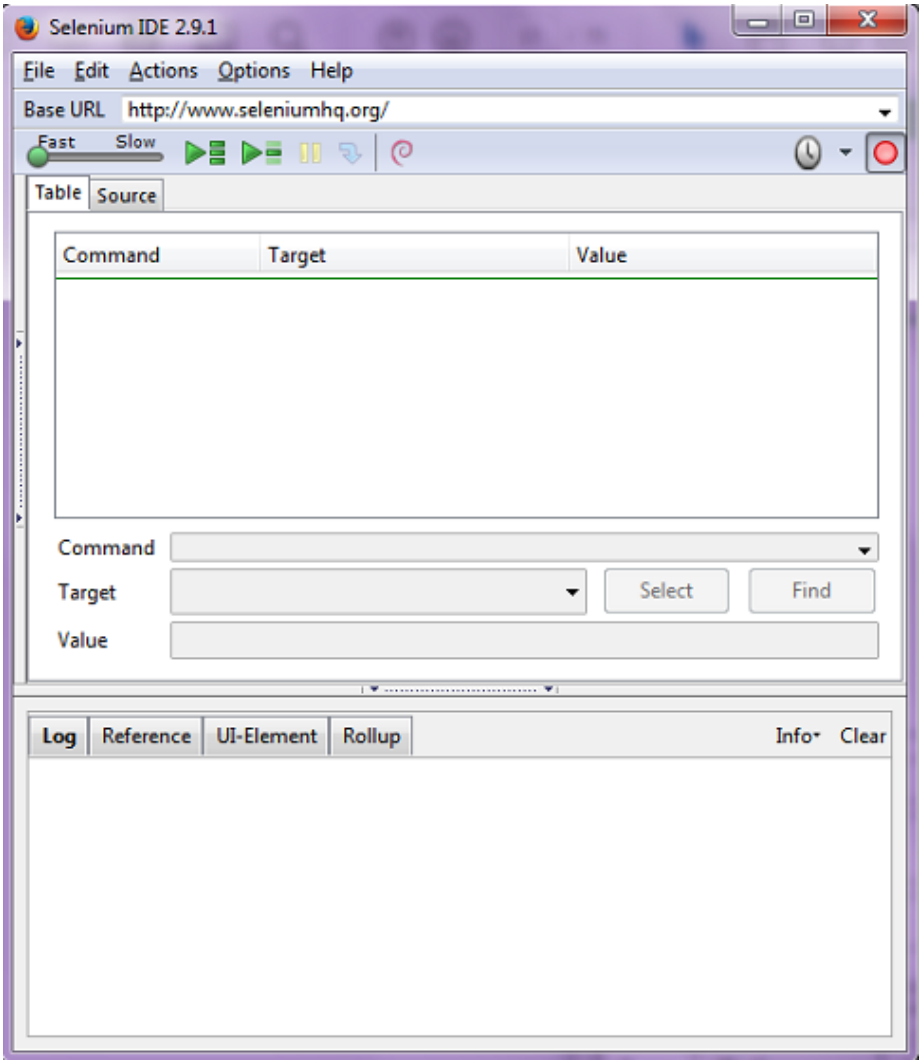
1. U Firefox pretraživaču otvoriti stranicu http://www.seleniumhq.org/download/
2. U sekciji Selenium IDE kliknuti na link from addons.mozilla.org
3. Na otvorenoj stranici kliknuti na dugme [+ Add to Firefox].
4. Otvoriće se iskačući prozor o instalaciji Selenium IDE dodatka. Kliknuti na dugme „Install”.
5. U novom prozoru koji iskače ispisana je poruka da će Selenium IDE biti instaliran posle ponovog pokretanja pretraživača. Klikom na dugme „Restart Now” pretraživač će se zatvoriti i ponovo otvoriti.

6. Selenium IDE dodatak je instaliran i dostupan u „Web Developer” sekciji Mozilla Firefox pretraživača. Takođe, postoji i ikonica na traci sa alatkama (prikazano na slici 2).

Slika 2: Selenium ikonica na traci sa alatkama

Opis Selenium IDE interfejsa

Jedan od načina da se pristupi Selenium IDE interfejsu je klikom na ikonicu na traci sa alatkama. Automatski se otvara prozor prikazan na slici 3:



Slika 3: Selenium IDE interfejs

„File” meni sadrži opcije za kreiranje novih test skriptova, za otvaranje postojećih, za čuvanje skriptova i za njihovo eksportovanje u jezik po izboru. Sve ove opcije su, takođe, dostupne i za grupu test skriptova (eng. Test Suite).

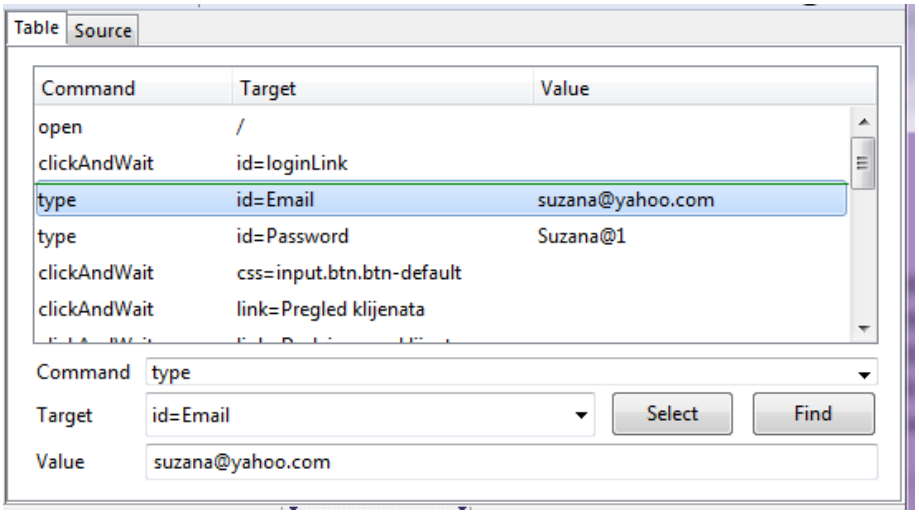
Opcije za izmenu test skriptova, nalaze se u „Edit” meniju. To su uobičajene opcije za izmenu teksta, kao što su kopiranje, brisanje, vraćanje izvršenih akcija unazad, koje se mogu primeniti i na test skriptove, a postoje i dve dodatne opcije – „Insert New Command” i „Insert New Comment” za dodavanje komandi i komentara prilikom ručnog pisanja test skriptova.

Snimanje i izvršavanje test skiptova kontroliše se opcijama menija „Actions”. Ovaj meni uključuje i opcije za debagovanje – „Toggle Breakpoint”, „Set/Clear Start Point” i „Step”. Test skriptovi mogu da se izvršavaju pojedinačno ili u grupi, jednom ili periodično. Takođe je moguće kontrolisati i brzinu izvršavanja test skriptova, stopiranje izvršavanja testa, proći kroz skript liniju po liniju. Gotovo sve pomenute akcije se mogu izvršiti i klikom na odgovarajuće dugme koje se nalazi na traci sa alatkama na istom prozoru.

Odabir formata u kome će definisani testovi biti čuvani vrši se u meniju „Options”.

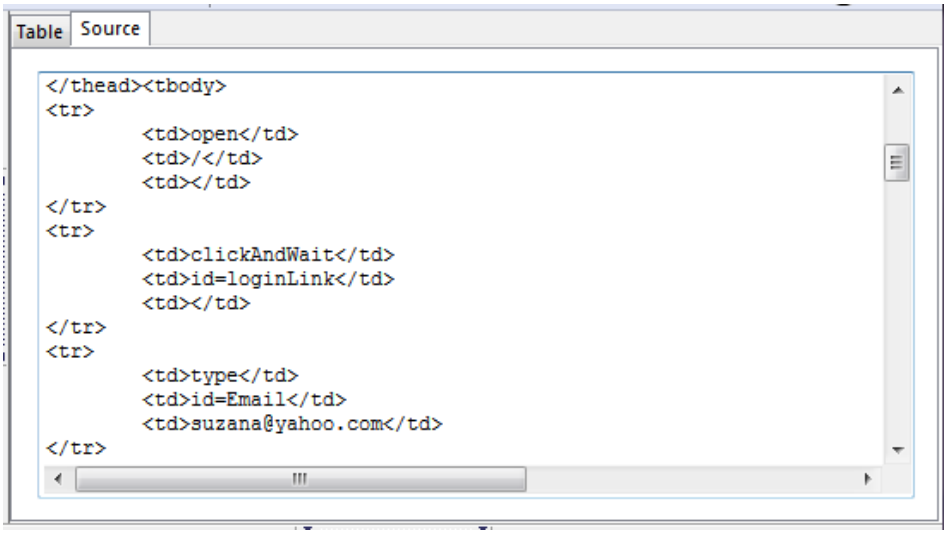
Test skriptovi se kreiraju i izvršavaju u prozoru sa karticama „Table” i „Source”.

Kartica „Table” ima tri kolone – „Command”, „Target” i „Value” koje opisuju pojedinačne komande i njihove parametre. Ove kolone, za izabranu komandu, imaju iste vrednosti kao i istoimena polja ispod prozora za prikaz skripta (prikaz na slici 4). „Command” polje sadrži komandu koju korisnik može da unese ili izabere iz liste dostupnih komandi. Polje „Target” opisuje lokaciju elementa na koji se odnosi komanda, a polje „Value” sadrži ulazne vrednosti korisnika za neke komande, kao što su komande za unos teksta.



Slika 4: Prikaz komandi korisnika uključujući i parametre

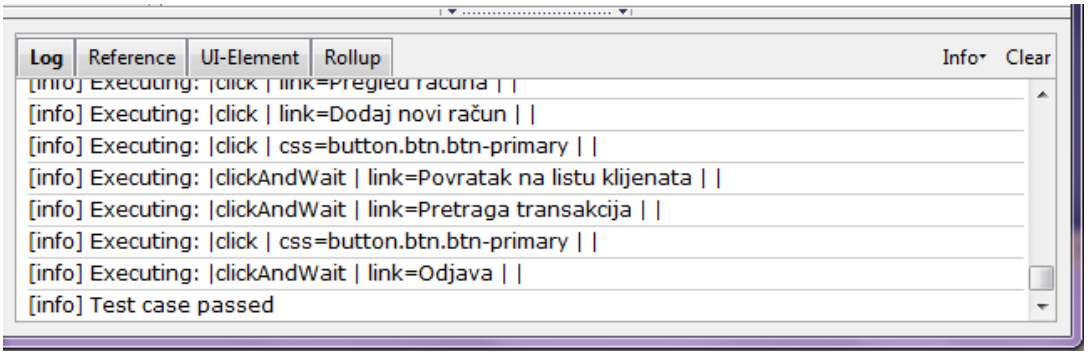
„Source” kartica prikazuje test skript u formatu u kome će biti sačuvan. Podrazumevani format je HTML format (slika 5), ali korisnik može da se opredeli da skript bude sačuvan u nekom od programskih jezika C#, Java, Python ili Ruby.



Slika 5: Prikaz komandi korisnika u HTML formatu

U donjem delu Selenium IDE interfejsa je prozor sa četiri kartice – „Log”, „Reference”, „UI-Element” i „Rollup”.

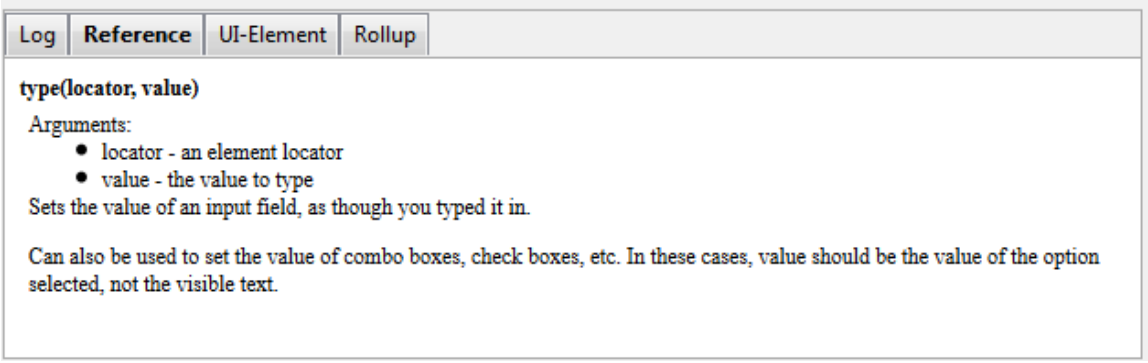
„Log” kartica ima svoju ulogu prilikom izvršavanja test skripta jer pokazuje status svake komande koja se izvršava i poruku o grešci, ukoliko do greške dođe, što je posebno važno za otklanjanje grešaka. Videti prikaz na slici 6.



Slika 6: Prikaz statusa izvršavanja komandi

„Reference” kartica sadrži dokumentaciju o Seleniz komandama. Svaki put kada se na „Table” kartici prozora za prikaz test skriptova menja neka komanda ili unosi nova, „Reference” kartica donjeg prozora je automatski izabrana jer ona sadrži opis svake komande, broj parametara, njihov redosled i tip. Prilikom pisanja ili modifikacija test skriptova mora se

voditi računa da vrednosti polja „Target” i „Value” za izabranu komandu odgovaraju opisu komande na „Reference” kartici (slika 7), inače komanda neće biti izvršena ispravno. Selenium komande po definiciji imaju dva parametra („Target” i „Value”). Međutim, veliki broj komandi ima jedan parametar, a takođe postoje i komande bez parametara.



Slika 7: Opis komande „type”

„UI-Element” i „Rollup” su rezervisane za naprednije upotrebe Selenium IDE-a. „UI-Element” kartica nalazi svoju primenu prilikom mapiranja između imena elemenata na Web stranici i samih elemenata, a „Rollup” za grupisanje više komandi u jednu.

Tipovi komandi u Selenium IDE-u

Selenium IDE raspolaže velikim brojem komandi koje podržavaju testiranje Web aplikacija sa različitih aspekata.

Test se izvršava na osnovu zadatih komandi, tačnije komande su te koje diktiraju izvršavanje testa. Sve komande Seleniza pripadaju jednoj od sledećih grupa:

- Komande akcija (eng. Actions)
- Komande pristupa (eng. Accessors)
- Komande potvrde (eng. Assertions)

Komande akcija su klik na link, klik na dugme, selektovanje vrednosti iz skupa vrednosti i mnoge druge. Ukoliko je akcija neuspešna, izvršavanje test skripta se stopira. Mnoge komande akcija imaju „AndWait” u svom imenu, kao što je to „ClickAndWait” komanda. Ovaj sufiks govori Selenium-u da izvršavanjem komande pretraživač upućuje poziv serveru i da Selenium treba da sačeka da se učita nova stranica.

Komande pristupa ispituju stanje aplikacije i čuvaju rezultat u promenljivama. Primer takve komande je „storeTitle” komanda.

Komande potvrde slične su komandama pristupa, s tim što one upoređuju stanje aplikacije sa očekivanim rezultatom, na primer, da li je naslov odgovarajući, da je neko polje popunjeno i slično.

Svaka komanda potvrde se može naći u jednom od tri oblika:

- „assert",
- „verify"
- „waitFor"

Ukoliko prilikom izvršavanja „assert" komande dođe do greške, izvršavanje test skripta se prekida. Međutim, ako prilikom izvršavanja „verify" komande dođe do greške, izvršavanje test skripta se nastavlja uz poruku o grešci. „Assert" komanda se najčešće koristi za proveru da li je neka stranica učitana i prekida se izvršavanje ako nije, dok se „verify" koristi za proveru prisustva elemenata na stranici jer ukoliko neki od elemenata nije pronađen, test nastavlja svoje izvršavanje i proveru ostalih polja na stranici.

Komanda „waitFor" čeka da neki uslov bude ispunjen. Uspešno se izvršava ako je uslov već ispunjen, a ne izvršava se ako uslov nije ispunjen u nekom određenom intervalu vremena.

Lociranje elemenata u Selenium IDE-u

Komande Selenium-a koje imaju parametar „Target", zahtevaju lokaciju elementa Web stranice na koji se odnose te komande**.** Vrednost parametra „Target" se obično označava kao lokator. Postoji veliki broj načina na osnovu kojih se vrši lociranje elemenata na određenoj stranici korišćenjem alata Selenium IDE ([8]):

- **Lociranje elemenata putem identifikatora** je najčešće korišćeni pristup. U ovom pristupu, biće lociran prvi element čiji se ID poklapa sa vrednošću lokatora. Ako se ID-evi ne poklapaju ni za jedan element, onda će biti lociran prvi element čije se ime atributa poklapa sa vrednošću lokatora.
- **Lociranje elemenata putem ID-a** se koristi onda kada je poznat ID elementa koji se locira. Lociranje elementa putem ID-a je dosta određenije od prethodnog pristupa jer se vrši dodeljivanje tačne vrednosti ID-a i na osnovu te vrednosti se vrši lociranje elementa.
- **Lociranje elemenata putem imena atributa –** slično prethodnom pristupu, locira se element čije ime atributa se poklapa sa vrednošću lokatora. Ukoliko postoji više elemenata sa istim imenom atributa, koristi se filter koji će dalje precizirati traženi element.
- **Lociranje elemenata putem Link teksta** – jednostavan način za lociranje linka na Web stranici korišćenjem teksta linka kao vrednosti lokatora. Ukoliko postoje dva linka sa istim imenom, prvi će biti lociran.

- **Lociranje elemenata putem XPatha –** XPath je jezik za lociranje elemenata u XML dokumentima. Kako se HTML može posmatrati kao implementacija XML-a, XPath se može koristiti u Selenium-u za lociranje elemenata na Web stranici. XPath je koristan u situacijama kada ne postoji odgovrajući ID ili ime atributa koji želimo da identifikujemo. Kako jedino XPath lokatori počinju sa „//”, nije potrebno uključiti xpath= label prilikom opisa XPath lokatora.
- **Lociranje elemenata putem DOM-a** – DOM (Document Object Model) je interfejs koji HTML, XHTML i XML dokumente tretira kao drvolike strukture u čijim se čvorovima nalaze objekti koji predstavljaju delove dokumenata. Za pristup DOM objektima u HTML-u koristi se JavaScript. Slično XPath lokatorima, koji jedini počinju sa „//”, DOM lokatori jedini počinju sa „document” i nije potrebno eksplicitno navoditi dom= label u specifikaciji lokatora.
- **Lociranje elemenata putem CSS-a –** CSS (Cascading Style Sheets) predstavlja jezik kojim se definišu stilovi Web stranica i izgled korisničkog interfejsa pisanog u HTML-u, XHTML-u i XML-u. Različiti selektori CSS-a mogu se koristiti u Selenium IDE-u za identifikaciju elemenata na stranici.

Podudaranje obrazaca teksta

Kod Seleniz komandi koje imaju drugi parameter, „Value”, vrednost tog parametra se obično može opisati nekim tekstualnim obrascem (eng. text pattern). Pri tom se pravi razlika između obrasca za verifikaciju i potvrdu sadržaja stranice (obično argument komandi „assert” i „verify”) i obrasca za unos teksta u neko polje ili izbor vrednosti iz liste vrednosti. Bez obzira na to da li se odnose na prvi ili na drugi slučaj, mogu pripadati „glob” obrascima (eng. globbing patterns), regularnim izrazima (eng. regular expressions) ili tačnim obrascima (eng. exact pattern) ([8]).

Glob obrasci se obično vezuju za DOS ili Linux komande jer se javljaju u obliku *.c pri čemu navedeni obrazac opisuje sve datoteke sa ekstenzijom .c. U Selenium-u se koriste specijalni karakteri * (opisuje jedan ili nijedan karakter ili više bilo kojih karaktera) i [] (opsuje bilo koji karakter naveden u uglastim zagradama). Obično se reč „glob” navodi ispred ovog tipa obrasca kad se koristi u komandi, ali je ovaj tip i podrazumevan, pa navođenje reči „glob” nije obavezno.

Regularni izrazi u Selenium-u se grade od skupa specijalnih karaktera koje podržava i JavaScript. Regularni izrazi u komandama moraju da imaju prefix regexp: (osetljiv na veličinu slova) ili regexpi: (nije osetljiv na veličinu slova).

Tačni obrasci ne koriste specijalne karaktere. Kod tačnih obrazaca je ‘*’ uvek ‘*’. Prefiks exact: se koristi pri upotrebi ovih obrazaca.

3.2.2 Selenium WebDriver

Selenium WebDriver prevazilazi ograničenja Selenium IDE-a u pogledu mogućnosti izvršavanja test skriptova u različitim pretraživačima, ali i u pogledu testiranja nekih funkcionalnosti kao što su postavljanje na stranicu ili preuzimanje dokumenata, zatim problemi vezani za rukovanje prozorima upozorenja prilikom potvrde određenih akcija i testiranje dinamičkih kompenti na Web stranici.

Selenium WebDriver koristi WebDriver API kao interfejs između koda u nekom od programskih jezika i drajvera za različite pretraživače. Programski jezici komuniciraju sa Selenium WebDriver API-jem i API je taj koji preuzima komande iz koda napisanog u nekom od programskih jezika, tumači preuzete komande i šalje ih odgovarujućem pretraživaču. API sadrži set zajedničkih biblioteka koje omogućavaju slanje komadi odgovarajućem drajveru koji upravlja samim pretraživačem. Selenium obezbeđuje drajvere za većinu pretraživača, kao što su Mozilla Firefox, Chrome, Safari, IE, Opera i mnogi dugi ([8]).

Pre nastanka Selenium WebDrivera, za izvršavanje test skriptova u različitim pretraživačima, koristio se Selenium Remote Control. Osnovna razlika između Remote Control-a i WebDriver-a jeste u načinu na koji oni komuniciraju sa pretraživačima. Selenium Remote Control je primenjivao isti princip za sve pretraživače – „ubacivao" je JavaScript funkcije u pretraživač kada se učita, a onda koristio svoj JavaScipt kôd da manipuliše aplikacijom u pretraživaču. Selenium WebDriver ima drugačiji pristup za svaki pretraživač jer koristi njihovu ugrađenu podršku za automatizaciju. Zato je izvršavanje testova korišćenjem WebDriver-a brže nego korišćenjem Remote Control-a ([8]).

Selenium WebDriver i Java

Da bi test skriptovi mogli da se izvršavaju korišćenjem Selenium WebDriver-a, potrebno je obezbediti odgovarajuće okruženje koje zavisi od jezika u kojem je napisan test skript. U praksi se Selenium WebDriver najčešće javlja u kombinaciji sa Java programskim jezikom, pa će osnovne tehnike koje koristi WebDriver biti opisane kroz Java kôd korišćenjem Eclipse-a i JUnit okvira za testiranje.

Okruženje se podešava na sledeći način ([11]):

1. Sa lokacije http://www.oracle.com/technetwork/java/javase/downloads/index.html preuzeti **JDK** (Java Software Development Kit) koji odgovara operativnom sistemu računara na kojem se vrši testiranje i instalirati Javu ukoliko već ne postoji na računaru.
2. Preuzeti **Eclipse for Java Developers** sa lokacije http://www.eclipse.org/downloads/: u vidu zip datoteke sa imenom „eclipse-jee-mars-1-win32-x86_64.zip".

3. Na željenoj lokaciji raspakovati zip datoteku i pokrenuti Eclipse klikom na izvršnu datoteku eclipse.exe.
4. Preuzeti **Selenium Java Client Driver** sa lokacije http://www.seleniumhq.org/download/ u obliku zip datoteke „selenium-java-3.0.0-beta3.zip".
5. Na željenoj lokaciji raspakovati zip datoteku. Direktorijum selenium-java-3.0.0-beta3 sadrži JAR datoteke sa bibliotekama potrebnim za korišćenje WebDriver-a
6. U Eclipse-u napraviti novi projekat i uvesti sve JAR datoteke pomenute u prethodnom koraku

Upravljanje stranicama i prozorima u Selenium WebDriver-u

Svaki test skript u Selenium WebDriver-u i Selenium IDE-u počinje od stranice koja se testira. Osnovna metoda za prelazak na željenu stranicu, koja se koristi u Selenium WebDriver-u, je get() metoda ([8]). Argument get() metode je URL stranice.

```
driver.get("http://www.google.com");
```

Osim get() metode, za prelazak na stranicu se koristi i navigate().to() metoda koja kao argument takođe prihvata URL zahtevane stranice. Get() i navigate().to() metode se primenjuju na potpuno isti način, razlika je samo u zapisu. Navigate() metoda javlja se u još dva oblika koja omogućavaju kretanje unapred i unazad kroz istoriju pretraživanja:

```
navigate().forward()
navigate().back()
```

Metoda switchTo() omogućava prelazak sa trenutne stranice na otvoreni okvir ili novi otvoreni prozor na trenutnoj stranici, kao i vraćanje sa novootvorenog prozora na trenutnu stranicu.

```
driver.switchTo().window(<naziv prozora>);
driver.switchTo().frame(<naziv okvira>);
```

Jedna od novina Selenium WebDriver-a u odnosu na prethodne verzije Selenium-a jeste podrška „iskačućim" prozorima (eng. pop-up window). Uz pomoć switchTo().alert() metode, može da se pristupi tekstu ispisanom na prozoru:

```
Alert alert = driver.switchTo().alert();
```

Ova naredba otvara iskačući prozor sa porukom, kojim sada može da se manipuliše – da se prihvati ili odbije upozorenje i da se čita ili menja tekst poruke.

Implicitno i eksplicitno čekanje

U zavisnosti od različitih faktora, uključujući kombinaciju operativnog sistema i pretraživača, WebDriver može čekati ili ne čekati da se u potpunonosti učita sadržaj neke stranice. Ukoliko se izvršavanje testa nastavi, a stranica nije u potpunosti učitana, doći će do greške u izvršavanju testa.

Zato je sinhronizacija između aplikacije, koja se testira i WebDrivera koji izvršava test script, veoma bitna. To posebno važi za komplikovane Web sajtove, aplikacije koje u velikoj meri koriste JavaScript, kao i za aplikacije koje se sastoje iz samo jedne Web stranice, a čije se promene u strukturi, zasnovane na akcijama korisnika, vrše preko JavaScripta. Nakon svakog otvaranja i učitavanja nove stranice u sistemu je potrebno obezbediti odgovarajuće čekanje dovoljno da stranica bude učitana i spremna za upotrebu ([8]).

Selenium WebDriver omogućuje sinhronizaciju na dva načina. To su sinhronizacija implicitnim čekanjem i sinhronizacija eksplicitnim čekanjem ([10]).

Sinhronizacija implicitnim čekanjem je ugrađena sinhronizacija u sam sistem. Ukoliko WebDriver ne može odmah da pronađe element na stranici, mora da sačeka neko određeno vreme da se element pojavi. To vreme određuje korisnik i važi globalno, to jest primenjuje se na pretragu svih elemenata. Podrazumevano vreme čekanja je 0 sekundi. Prilikom postavljanja vremena potrebnog za čekanje treba biti pažljiv, posebno ukoliko aplikacija ima normalan odziv, jer implicitno čekanje usporava izvršavanje testova.

Selenium WebDriver obezbeđuje implicitno čekanje uz pomoć Timeouts interfejsa i implicitlyWait() metode. U sledećem primeru, test će čekati 10 sekundi da se element pojavi u DOM strukturi:

```
driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
```

Kod eksplicitnog čekanja, korisnik definiše odgovarajući uslov čije izvršenje treba da se sačeka i tek nakon toga se vrši željena akcija na učitanoj stranici. Eksplicitno čekanje se primenjuje samo u onim slučajevima kada se očekuje da će trebati više vremena da se neki element učita, dok ostatak aplikacije radi normalno.

Selenium WebDriver, uz pomoć WebDriverWait and ExpectedCondition klasa, podržava sinhronizaciju eksplicitnim čekanjem. ExpectedCondition klasa sadrži skup predefinisanih uslova za čekanje pre nego što se nastavi sa izvršavanjem testa.

U sledećem primeru, pomoću statičke metode ElementIsVisible(), vrši se čekanje da odgovarajući link bude vidljiv na određenoj Web stranici.

```
WebDriverWait wait = new WebDriverWait(driver, 10);
```

```
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpa
th(<xpath niska>)));
```

Lociranje elemenata u Selenium WebDriver-u

Slično Selenium IDE-u, SeleniumWebDriver takođe koristi različite metode za lociranje elemenata na Web stranici. Međutim, pre pisanja samog koda, potrebno je pažljivo analizirati samu stranicu i elemente kako bi se razumela njihova struktura u aplikaciji, na primer, koji su atributi definisani za određene elemente, kakvi su JavaScript i Ajax pozivi napravljeni iz aplikacije i slično.

Pretraživač renderuje elemente u aplikaciji, sakrivajući HTML kôd za krajnje korisnike. Zato se koriste različiti alati koji olakšavaju proučavanje koda u pozadini. Neki od njih su:

- Firebug za Firefox
- Google Developer Tools za Chrome
- Web Inspector za Safari

To su takozvani Browser Developer alati koji su ugrađeni u pretraživače i pružaju informacije na osnovu kojih se vrši lociranje elemenata, kao i informacije o njihovim atributima, JavaScript pozivima, stilovima atributa, izvorima stranica i slično. HTML pozadina Web stranice se otvara klikom tastera „F12" na tastaturi ili ako se izabere neki element na Web stranici, klikne desnim klikom miša i u prikazanoj listi odabere „Inspect Element" ([10]).

Za lociranje elemenata u Selenium WebDriver-u koriste se dve metode obezbeđene od strane WebDriver i WebElement klase. To su findElement() i findElements() metode.

Metoda findElement() vraća prvi element Web stranice (objekat klase WebElement) koji zadovoljava kriterijum na osnovu kojeg se vrši pretraga. Ukoliko ne uspe da nađe element koji odgovara kriterijumu pretrage, metoda findElement() baca izuzetak NoSuchElementFindException.

Metoda findElements() vraća listu Web elemenata koji zadovoljavaju kriterijum lociranja elementa. Ukoliko ne postoje takvi elmenti, vraća praznu listu. Ova metoda je jako korisna kada se radi sa grupom sličnih elemenata. Na primer, možemo dobiti sve linkove prikazane na stranici, sve redove iz tabele ili sve vrednosti iz padajuće liste.

Selenium WebDriver obezbeđuje By klasu, čiji se objekat koristi kao argument „find" metode, kako bi podržao različite načine lociranja elemenata na Web stranici ([9] i [10]). To su:

- **Lociranje elemenata putem ID-a** – najjednostavniji, najbrži i najefikasniji način za lociranje elemenata na Web stranici. U poređenju sa tekstom, skriptama koje koriste, ID-

evi su najmanje skloni promenama u aplikaciji, zato je ovaj način lociranja elemenata i najzastupljeniji. Da bi lociranje elemenata ovom metodom bilo uspešno, potrebno je da različiti elementi imaju različite ID-eve, to jest da su ID-evi jedinstveno određeni.

Poziv metode findElement(), u slučaju pretrage po ID-u, izgleda:

```
WebElement element = driver.findElement(By.id(<ID elementa>))
```

- **Lociranje elemenata putem imena atributa** – ime atributa je najčešće korišćeno za lociranje elemenata kao što su tekst polja ili radio dugmad. Kao i kod ID-a, mala je verovatnoća promene imena atributa. Ime atributa ne mora da bude jedinstveno, te, u slučaju da postoji više elemenata sa istim imenom, metoda vraća prvi element na strani sa definisanom vrednošću, što možda i nije element koji se zahteva. To može da izazove prekid izvršavanja testa.

```
WebElement element = driver.findElement(By.name(<vrednost name atributa>))
```

- **Lociranje elemenata putem imena klase** – Uloga atributa class u HTML dokumentima je da omogući primenu CSS-a na odgovarajući element, ali se takođe se može koristiti i za identifikovanje elemenata.

```
WebElement element = driver.findElement(By.className(<vrednost class atributa>))
```

- **Lociranje elemenata putem imena etikete –** By klasa Selenium WebDriver-a ima metodu tagName() za lociranje elemenata na osnovu njihove HTML etikete. Ova metoda je veoma korisna kada je potrebno locirati na primer, sve <tr> etikete u tabeli.

  Sledeći segment koda prikazuje na koji način se mogu izbrojati redovi u tabeli:

```
WebElement tabela = driver.findElement(By.id("Tabela"));
List<WebElement> redovi = table.findElements(By.tagName("tr"));
assertEquals(10, redovi.size());
```

- **Lociranje elemenata putem Link teksta –** linkText() metoda By klase omogućuje lociranje linkova na Web stranici korišćenjem njihovog teksta. Javlja se u obliku:

```
WebElement Link = driver.findElement(By.linkText("Link"));
```

- **Lociranje elemenata putem nepotpunog Link teksta –** Klasa By ima još i metodu partialLinkText(). Koristi se za lociranje linkova sa tekstom koji se dinamički menja, to jest, tekstom koji može biti drugačiji prilikom različitih poseta jednoj istoj stranici. U takvim situacijama koristi se zajednički, nepromenljivi deo teksta linka.

Jedan od primera jeste link „Primljene Poruke” kod različitih aplikacija za razmenu poruka. Ovaj link se menja dinamički jer prikazuje broj primljenih poruka. Za argument partialLinkText() metode uzima se deo „Primljene Poruke” jer se on ne menja.

```
WebElement porukeLink =
driver.findElement(By.partialLinkText("Primljene Poruke"));
```

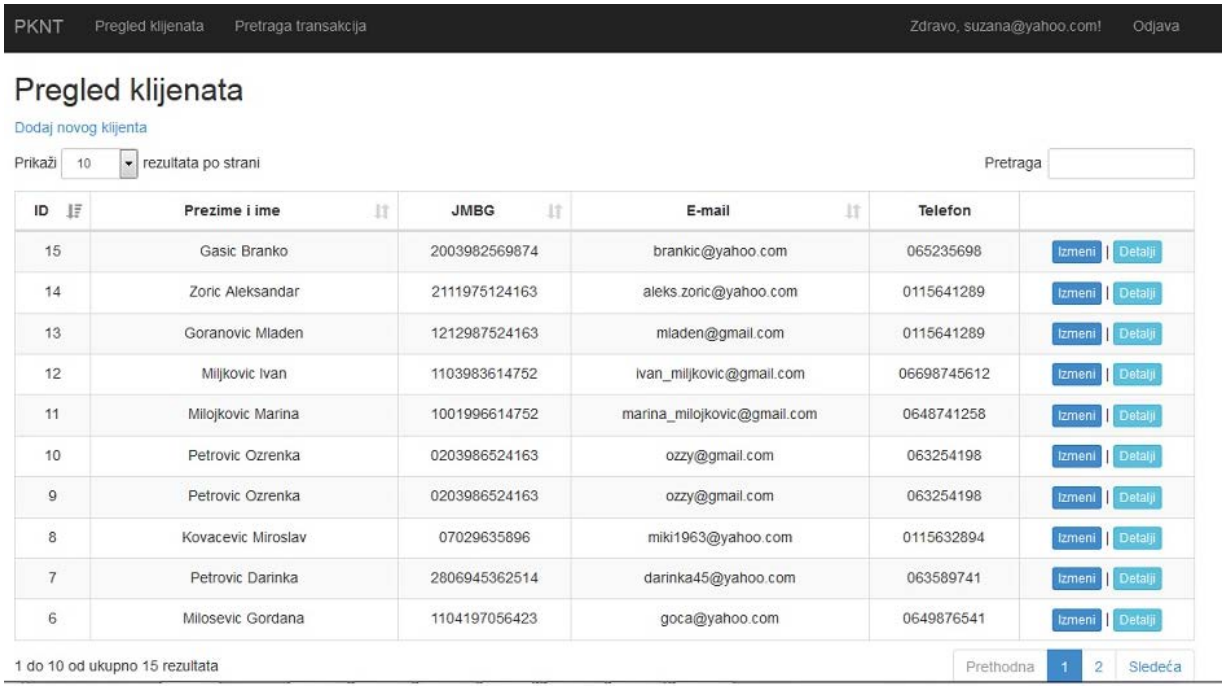
- **Lociranje elemenata putem CSS selektora –** CSS selektor je šablon i deo CSS pravila koji se poklapa sa skupom elemenata u HTML dokumentu. CSS selektori raspolažu velikim brojem metoda, pravila i šablona za lociranje elemenata na Web stranici, koje i Selenium WebDriver koristi. Neki od njih su: lociranje putem apsolutne putanje, lociranje putem relativne putanje, putem ID selektora, imena atributa i tako dalje. Lociranje elemenata putem CSS selektora je brže i pouzdanije u odnosu na lociranje putem XPath lokatora.
- **Lociranje elemenata putem XPath-a –** XPath predstavlja „drvoliku” reprezentaciju XML dokumenta i omogućava navigaciju kroz drvo izborom čvorova po različitim kriterijumima**.** Selenium WebDriver podržava XPath za lociranje elemenata korišćenjem XPath izraza ili upite. Međutim, ova metoda za lociranje elemenata je jako spora i zbog toga ne baš pogodna za primenu.
  Važna razlika između ove i metode lociranja elemenata putem CSS selektora je što, korišćenjem XPath-a, mogu da se lociraju elementi penjući se i spuštajući se u hijerarhiji elemenata, dok CSS selektor dopušta samo spuštanje kroz hijerarhiju. To znači da XPath metodom, možemo da lociramo „roditeljski” element polazeći od „dete” elementa.
  XPath, kao i CSS, takođe raspolaže velikim brojem načina za lociranje elemenata: putem apsolutne putanje, relativne putanje, ID selektora, atributa i drugih.

4 Automatizovano testiranje Web aplikacije „Pregled klijenata i novčanih transakcija - PKNT”

4.1 Opis aplikacije „PKNT”

Na primeru Web aplikacije „PKNT” biće ilustrovano automatizovano testiranje korišćenjem Selenium alata. Aplikacija je namenjena bankarskim savetnicima i sadrži informacije o klijentima savetnika i njihovim novčanim transakcijama. Svaki savetnik ima pristup aplikaciji i vidi samo svoje klijente i njihova novčana sredstva. Savetnik aplikaciji pristupa preko stranice za prijavu unoseći svoj e-mail i lozinku. Po prijavi, korisnik može da odabere jednu od dve kartice – karticu „Pregled klijenata” ili karticu „Pretraga transakcija”.

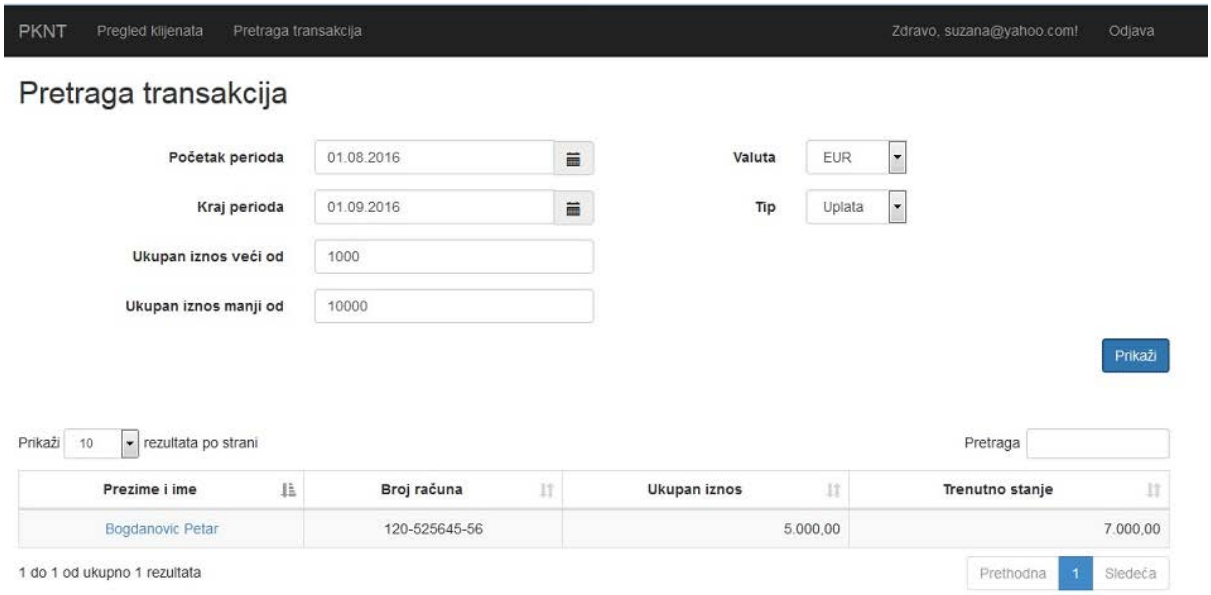
Na stranici „Pregled klijenata” prikazana je tabela sa osnovnim podacima o klijentima, kao što su prezime i ime, JMBG, e-mail adresa i broj telefona (videti sliku 8). Detalji u tabeli mogu da se urede po bilo kojoj koloni, a podrazumevani redosled redova jeste opadajući po prvoj koloni. Korišćenjem padajuće liste u gornjem levom uglu, iznad tabele, korisnik postavlja željeni broj redova tabele na stranici, koji može biti 5, 10, 25, 50 ili 100. Takođe, korisnik može da pretražuje tabelu po imenu i/ili prezimenu klijenta, po JMBG-u, e-mail adresi ili broju telefona.



Slika 8: Stranica sa informacijama o klijentima

Za svakog klijenta na raspolaganju su dva dugmeta – „Izmeni” i „Detalji” kojima se otvaraju stranice „Izmena klijenta” i „Detalji klijenta”. Stranica „Izmena klijenta” sadrži formu koja omogućava izmene podataka o klijentima i čuvanje tih izmena, dok stranica „Detalji klijenta” prikazuje detaljnije informacije o izabranom klijentu, uključujući i pregled računa tog klijenta klikom na link „Pregled računa”. Korisnik može i da doda novog klijenta – na stanici „Pregled klijenata” dostupan je link „Dodaj novog klijenta” koji otvara stranicu sa praznom formom gde je moguće uneti informacije o novom klijentu i sačuvati formu. Po čuvanju forme, novi klijent bi trebalo da se prikaže u tabeli na stranici „Pregled klijenata”.

Stranica „Pretraga transakcija”, za izabrani period i valutu, prikazuje iznos uplata ili isplata po računima klijenata. Valuta može biti „RSD” ili „EUR” s obzirom na to da klijenti imaju dinarske ili devizne račune. Korisnik može da odredi i opseg u kome želi da vidi iznos uplata ili isplata, unoseći vrednosti u polja „Ukupan iznos veći od” i „Ukupan iznos manji od”. Kada su svi uslovi izbrani – vremenski period, valuta (RSD ili EUR), tip (Uplata ili Isplata) i željeni opseg ukupnog iznosa, klikom na dugme „Prikaži”, prikazuje se tabela sa kolonama „Prezime i ime”, „Broj računa”, „Iznos” i „Trenutno stanje”. Vrednosti u tabeli mogu se urediti po bilo kojoj koloni i, kao i u slučaju tabele na stranici „Pregled klijenata”, moguće je prikazati rezultat u 5, 10, 25, 50 ili 100 redova na stranici. Dostupna je i pretraga rezultata unosom u polje „Pretraga” imena i/ili prezimena klijenta i broja računa (videti sliku 9).



Slika 9: Stranica sa informacijama o transakcijama

4.2 Plan za testiranje „PKNT” aplikacije

Aplikacija „PKNT” će biti funkcionalno testirana i sa tačke gledišta testera – dakle, biće testirana na nivou sistemskog testiranja. Pri tom će se koristiti strategija sive kutije, s obzirom na to da se za pisanje automatizovanih testova koriste delovi koda iz HTML pozadine aplikacije. Za testiranje će biti primenjeni Selenium IDE i Selenium WebDriver alati, a testovi će se izvršavati u Mozilla Firefox pretraživaču, jer je to jedini pretraživač u kome funkcionišu oba ova alata.

Testiranje će biti podeljeno u dva dela:

1) Provera osnovnih funkcionalnosti - Svaki put kada se kreće sa testiranjem nekog dela aplikacije, pre detaljne provere svih funkcionalnosti, treba izvršiti osnovne provere (eng. Smoke Test) da bi se procenilo ima li uopšte smisla detaljnije testirati. Provera osnovnih funkcionalnosti aplikacije „PKNT” biće izvršena korišćenjem Selenium IDE alata.

2) Detaljno testiranje funkcionalnosti – Za detaljniju proveru funkcionalnosti aplikacije biće korišćen Selenium WebDriver alat jer su mogućnosti Selenium WebDriver-a mnogo šire nego mogućnosti Selenium IDE-a.

4.3 Provera osnovnih funkcionalnosti

Provera osnovnih funkcionalnosti „PKNT” aplikacije podrazumeva:

1) Proveru pristupa aplikaciji,
2) Proveru dostupnosti stranica „Pregled klijenata” i „Pretraga transakcija”,
3) Proveru dostupnosti stranica „Dodaj novog klijenta”, „Izmena klijenta” i „Detalji klijenta”,
4) Proveru prisustva elemenata na svim stranicama
5) Proveru funkcionalnosti osnovnih elemenata
6) Proveru odjave iz aplikacije

Uobičajena je praksa da se sve ove funkcionalnosti proveravaju jednim test skriptom.

Test skript se u Selenium IDE-u kreira na jednostavan način. Pritiskom na crveno dugme, započinje snimanje akcija korisnika, dok se, u isto vreme, odgovarajuće komande upisuju u prozor za pisanje i izvršavanje test skriptova. Akcije koje će se upisivati su klik na link, unos vrednosti u odgovarajuća polja, odabir željene opcije iz liste, selektovanje dugmadi za obeležavanje ili radio dugmadi i druge. Selenium IDE će predvideti koja komanda, sa

kojim identifikatorom, odgovara selektovanom UI elementu na Web stranici i vrednosti iste će biti ispisane u „Table” prozoru u poljima „Target” i „Value”.

Snimanje počinje od stranice čiji je URL naveden u „Base URL” polju u Selenium IDE-u. U slučaju „PKNT” aplikacije, base URL je http://localhost:46003/FA_Dim. Na slici 10 je prikazan test skript dobijen snimanjem. Kolone tabele odgovaraju kolonama „Command”, „Target” i „Value” u Selenium IDE-u.

Komanda **open** se odnosi na otvaranje stranice čiji je URL naveden u „Base URL” polju. Za njom sledi komanda **clickAndWait**, koja predstavlja klik na element čiji je id „loginLink”, a to je dugme za prijavu. Naredne dve komande **type** omogućavaju unos e-mail adrese i lozike u polja za prijavu u aplikaciju, a za njima sledi klik na dugme za prijavu. Preostali koraci u test skriptu su kombinacija **click** i **clickAndWait** komandi i služe za proveru dostupnosti stranica u aplikaciji putem linka ili dugmadi. Obe komande predstavljaju klik na neki element, ali se kod **clickAndWait** komande čeka da se stranica učita pre nego što se pređe na sledeću komandu. Ukoliko se ne sačeka da se stranica sa svim svojim elementima u potpunosti učita i krene u izvršavanje komandi koje se odnose na elemente te stranice, doći će do prekida izvršavanja testa.

SmokeTest		
open	/	
clickAndWait	id=loginLink	
type	id=Email	suzana@yahoo.com
type	id=Password	Suzana@1
clickAndWait	css=input.btn.btn-default	
clickAndWait	link=Pregled klijenata	
clickAndWait	link=Dodaj novog klijenta	
clickAndWait	link=Povratak na listu klijenata	
clickAndWait	link=Izmeni	
clickAndWait	link=Povratak na listu klijenata	
clickAndWait	link=Detalji	
click	link=Pregled računa	
click	link=Dodaj novi račun	
click	css=button.btn.btn-primary	
clickAndWait	link=Povratak na listu klijenata	
clickAndWait	link=Pretraga transakcija	
click	css=button.btn.btn-primary	
clickAndWait	link=Odjava	

Slika 10: Test skript za proveru osnovnih funkcionalnosti dobijen snimanjem

Ako pokrenemo izvršavanje testa, videćemo da će se on uspešno izvršiti. Međutim, da bismo ovaj test mogli nazvati pravim testom provere osnovnih funkcionalnosti, moramo dodati komande koje verifikuju prisustvo elemenata na stranicama jer je to neophodno za testiranje dataljnijih funkcionalnosti. Takođe će biti dodate komande za testiranje funkcionalnosti dugmadi za dodavanje i izmene klijenata i njihovih računa.

Za proveru prisustva elemenata biće korišćene komande **AssertElementPresent** i **VerifyElementPresent**. Razlika među njima je što „Assert" komande prekidaju izvršavanje testa ukoliko element nije pronađen, dok „Verify" komande, u istom slučaju, omogućavaju nastavak izvršavanja testa. **AssertElementPresent** komandu ima smisla koristiti prilikom provere prisustva elemenata na stranici za prijavu korisnika, jer ukoliko neki od elemenata nije prisutan, korisnik neće moći da pristupi aplikaciji i izvršavanje testa se mora prekinuti. Sličan princip se koristi i prilikom provere prisustva linkova ka nekim stranicama jer nema smisla vršiti testiranje na stranici ako stranica nije ni dostupna. Prisustvo ostalih elemenata na kojima se neće izvršavati dodatne akcije se proverava komandom **VerifyElementPresent**. Naredna slika (slika 11) prikazuje korake test skripta posle dodavanja komandi za stranice za prijavu, „Pregled klijenata" i „Dodavanje klijenta".

SmokeTest1		
open	/	
clickAndWait	id=loginLink	
assertElementPresent	id=Email	
type	id=Email	suzana@yahoo.com
assertElementPresent	id=Password	
type	id=Password	Suzana@1
assertElementPresent	css=input.btn.btn-default	
verifyElementPresent	link=Registruj se kao novi korisnik	
clickAndWait	css=input.btn.btn-default	
assertElementPresent	link=Pregled klijenata	
clickAndWait	link=Pregled klijenata	
assertElementPresent	id=fa-dim-table	
verifyElementPresent	css=input.form-control.input-sm	
verifyElementPresent	name=fa-dim-table_length	
assertElementPresent	link=Dodaj novog klijenta	
clickAndWait	link=Dodaj novog klijenta	
verifyElementPresent	id=FA_Full_Name	
verifyElementPresent	id=FA_Birth_Date	
verifyElementPresent	id=FA_JMBG	
verifyElementPresent	id=FA_Email	
verifyElementPresent	id=FA_Address	
verifyElementPresent	id=FA_Phone	
assertElementPresent	id=submit-button	
click	id=submit-button	

Slika 11: Test skript za proveru osnovnih funkcionalnosti sa dodatnim komandama za proveru prisustva elemenata

Ovaj test skript može da se sačuva u formatu nekog od podržanih programskih jezika, kao što su Java, C#, Python i Ruby. Dovoljno je samo u „File” meniju izabrati opciju „Export Test Case As” i potom odabrati željeni programski jezik.

Deo koda koji sledi prikazuje zapis Selenium komandi u programskom jeziku Java. To su komande za proveru prisustva elemenata i proveru prijave korisnika na početnoj stranici aplikacije.

```
@Test
public void testSmokeTest1() throws Exception {
  // open | / |
  driver.get(baseUrl + "/");

  // click | id=loginLink |
  driver.findElement(By.id("loginLink")).click();

  // assertElementPresent | id=Email |
  assertTrue(isElementPresent(By.id("Email")));

  // type | id=Email | suzana@yahoo.com
  driver.findElement(By.id("Email")).clear();
  driver.findElement(By.id("Email")).sendKeys("suzana@yahoo.com");

  // assertElementPresent | id=Password |
  assertTrue(isElementPresent(By.id("Password")));

  // type | id=Password | Suzana@1
  driver.findElement(By.id("Password")).clear();
  driver.findElement(By.id("Password")).sendKeys("Suzana@1");

  // assertElementPresent | css=input.btn.btn-default |
  assertTrue(isElementPresent(By.cssSelector("input.btn.btn-default")));

  // verifyElementPresent | link=Registruj se kao novi korisnik |
  try {
    assertTrue(isElementPresent(By.linkText("Registruj se kao novi korisnik")));
  } catch (Error e) {
    verificationErrors.append(e.toString());
```

Test provere osnovnih funkcionalnosti, preveden na neki programski jezik, može da se izvršava korišćenjem odgovarajućih alata za kompajliranje i izvršavanje takvih programa. Na primer, za programski jezik Java, može se koristiti Eclipse sa dodatkom biblioteka Selenium WebDrivera. Uz manje prilagođavanje test skripta okruženju, rezultat izvršavanja testa bi, takođe, trebalo da bude uspešan.

4.4 Detaljno testiranje funkcionalnosti

Organizacija testova u JUnit-u

Termin „Test Fixture” koristi se da označi klasu koja sadrži skup definisanih testova. Opciono može sadržati i **setUp()** i **tearDown()** metode. Iako se, u ovom radu, automatizovano testiranje ilustruje kroz JUnit, Test Fixture nije specifičnost samo ovog alata, već se koristi i u drugim alatima za testiranje.

```
@Before
public void setUp() throws Exception {
  System.setProperty("webdriver.firefox.bin", "D:\\Master\\FirefoxPortable\\firefoxportable.exe");
  driver = new FirefoxDriver();
  baseUrl = "http://localhost:46003/FA_Dim";
  driver.manage().timeouts().implicitlyWait(40, TimeUnit.SECONDS);
  wait = new WebDriverWait(driver, 10);

  driver.get(baseUrl);

  driver.findElement(By.xpath(".//*[@id='Email']")).sendKeys("suzana@yahoo.com");
  driver.findElement(By.xpath(".//*[@id='Password']")).sendKeys("Suzana@1");

  driver.findElement(By.xpath(".//*[@id='loginForm']/form/div[4]/div/input")).click();
}
```

Obavezna je @Test anotacija za svaku test metodu, ali i @Before za **setUp()** i @After za **tearDown()** metodu.

SetUp() metoda se uvek poziva pre test metoda jer je uobičajeno da ona sadrži početne postavke neophodne za izvršavanje testova, kao što su postavljanje Web adrese u pretraživaču na URL aplikacije koja se testira ili prijavljivanje na sistem. Sledi primer setUp() metode koja se koristi prilikom testiranja aplikacije „PKNT”.

Test metode, po pravilu, nemaju povratnu vrednost, niti parametre. Definišu se na sledeći način:

public void testNazivMetode();

Ukoliko se ne ispoštuje pravilo za odgovarajući potpis, izvrsavanje test metode neće biti pokrenuto.

tearDown() metoda sadrži akcije koje se izvršavaju nakon izvršavanja testova. Na primer, oslobađanje nekih resursa ili zatvaranje aplikacije.

```
@After
public void tearDown() throws Exception {
  driver.quit();
  String verificationErrorString = verificationErrors.toString();
  if (!"".equals(verificationErrorString)) {
    fail(verificationErrorString);
  }
}
```

Kako su podešavanja okruženja neophodna za izvršavanje svakog testa koji je deo Test Fixture-a, **setUp()** metoda će se pozivati pre poziva svake test metode, a **tearDown()** metoda posle poziva svake test metode. Dakle, ako Test Fixture ima dve test metode (**test1()** i **test2()**) i **setUp()** i **tearDown()** metode, redosled izvršavanja je sledeći:

1. @Before setUp()
2. @Test test1()
3. @After tearDown()
4. @Before setUp()
5. @Test test2()
6. @After tearDown()

Ukoliko je neophodno da se samo jednom izvrši inicijalizacija postavki sistema, pre izvršavanja testova i uklanjanje posle izvršavanja, postoji varijanta **setUp()** i **tearDown()** metode, koje se za to koriste. To su **setUpClass()** i **tearDownClass()** sa anotacijama @BeforeClass i @AfterClass. Tada je redosled izvršavanja:

1. @Before setUp()
2. @Test test1()
3. @Test test2()
4. @After tearDown()

Stranica „Pregled klijenata”

Detaljniji testovi na stranici „Pregled klijenata” uključuju:

1) Validaciju naslova
2) Validaciju vrednosti u padajućoj listi
3) Validaciju broja kolona u tabeli i broja vrsta u zavisnosti od izabrane vrednosti u padajućoj listi iznad tabele
4) Validaciju naslova kolona tabele
5) Validaciju formata vrednosti u tabeli

6) Validaciju uređenja vrednosti u tabeli
7) Validaciju polja za pretragu

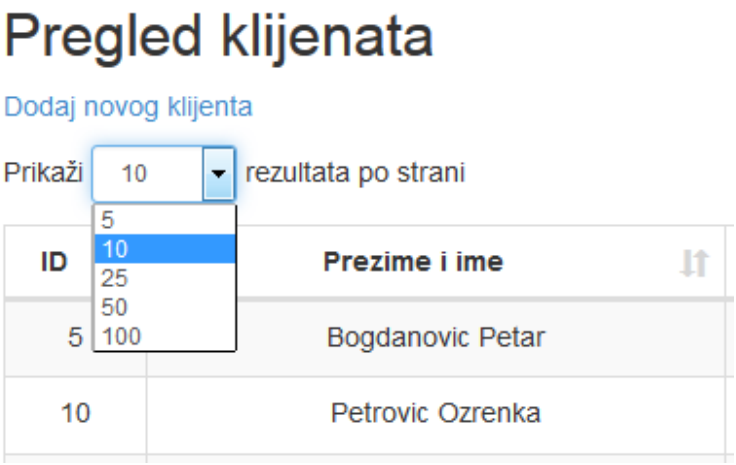
Sledi opis svakog od ovih testova:

1) Za validaciju naslova koristi se jednostavna funkcija koja poziva funkciju **assertEquals** klase **Assert**. **AssertEquals** poredi očekivanu vrednost (prvi argument funkcije) i stvarnu vrednost teksta elementa koji je identifikovan uz pomoć CSS selektora (drugi argument funkcije).

```
@Test
  public void testTitle() throws Exception {
      try {
          assertEquals("Pregled klijenata", driver.findElement(By.cssSelector("h2")).getText());
        } catch (Error e) {
          verificationErrors.append(e.toString()); }
  }
```

Ukoliko su dve vrednosti jednake, izvršavanje testa biće uspešno. Ukoliko nisu, funkcija izbacuje izuzetak. Ako se **assertEquals** funkcija koristi sa drugim funkcijama u okviru testa i rezultat poređenja nije tačan, izvršavanje testa će se prekinuti. U takvim situacijama, ako je poželjno samo porediti vrednosti, ali ne i prekinuti izvršavanje programa ako vrednosti nisu jednake assertEquals funkcija se stavlja u **try-catch** blok.

2) Podrazumevana vrednost padajuće liste za broj redova tabele po stranici je 10, a vrednosti su 5, 10, 25, 50 i 100 (videti sliku 12).



Slika 12: Izbor željenog broja redova tabele po jednoj stranici

Naredni test validira podrazumevanu vrednost i sve vrednosti u listi:

```
@Test
public void testDropDown() throws Exception

{


    String[] values = {"5", "10", "25", "50", "100"};
    Select dropDown = new Select(driver.findElement(
            By.xpath(".//*[@id='fa-dim-table_length']/label/select")));
    WebElement defaultOption  = dropDown.getFirstSelectedOption();
    String optionText = defaultOption.getText();
    System.out.println(optionText);
    Assert.assertEquals("10", optionText);
    List<WebElement> options = dropDown.getOptions();
    for (WebElement option : options) {
        boolean match = false;
        for(int i = 0; i < values.length; i++ ){
             if (option.getText().equals(values[i])){
                    match = true;
        }
        }
        Assert.assertTrue(match);
    }
}
```

U Selenium WebDriver-u postoji klasa **Select** za rad sa listama i padajućim listama. U testu se promenljivoj **dropDown** dodeljuje referenca na novi objekat klase Select koji je inicijalizovan objektom klase **WebElement**, a to je zapravo element sa xpath-om "*.//*[@id='fa-dim-table_length']/label/select*". Klasa **Select** raspolaže funkcijom **getFirstSelectedOption()** koja uvek vraća element koji je selektovan pa je pogodna za validaciju podrazumevane vrednosti liste.

Za validaciju svih vrednosti liste korišćena je metoda getOptions(), koja je , takođe, metoda iz klase **Select** koja vraća listu svih opcija – **options.** Dalje se u for petlji porede vrednosti liste **options** i niza niski **values** metodom **equals()** klase **String.**

3) Broj kolona u tabeli se određuje kao veličina liste koja sadrži polja prvog reda tabele.

```
@Test
public void testColumnCount()
{
    int columnCount = driver.findElements(
            By.xpath(".//*[@id='fa-dim-table']/tbody/tr[1]/td")).size();
    System.out.println("Broj kolona je " + columnCount);
    try{
    assertEquals(6, columnCount);
    } catch (Error e) {
          verificationErrors.append(e.toString());
        }
}
```

Brojanje redova u tabeli zavisi od postavljene željene vrednosti o broju redova po stranici. Sledeći test za svaku vrednost padajuće liste proverava broj redova u tabeli.

```
@Test
public void testRowCount()
{
    String[] values = {"5", "10", "25", "50", "100"};
    Select dropDown = new Select(driver.findElement(
            By.xpath(".//*[@id='fa-dim-table_length']/label/select")));

for (int i = 0; i < values.length; i++) {
    dropDown.selectByValue(values[i]);
    int rowCount1 = driver.findElements(By.xpath(".//*[@id='fa-dim-table']/tbody/tr")).size();
    System.out.println("Broj redova na prvoj stranici je " + rowCount1);
    assertTrue(rowCount1 == Integer.parseInt(values[i]) || rowCount1 < Integer.parseInt(values[i]));

    while (!HasClass(driver.findElement(By.id("fa-dim-table_next")), "disabled")) {
        driver.findElement(By.xpath(".//*[@id='fa-dim-table_next']/a")).click();
        int rowCount2 = driver.findElements(By.xpath(".//*[@id='fa-dim-table']/tbody/tr")).size();
        System.out.println("Broj redova na sledecoj stranici" + rowCount2);
        assertTrue(rowCount2 == Integer.parseInt(values[i]) || rowCount2 < Integer.parseInt(values[i]));
    }
    driver.findElement(By.xpath(".//*[@id='fa-dim-table_paginate']/ul/li[2]/a")).click();
}
}
```

Vrednosti padajuće liste se čuvaju u nizu **values.** U **for** petlji se svaki element niza selektuje metodom **selectByValue()** klase **Select.** Za selektovanu vrednost se broje redovi na prvoj stranici tabele i upoređuje broj redova sa izabranim brojem. Broj redova može biti jednak izabranom broju ili manji. Pošto je dopušteno da tabela ima više stranica, u slučaju da je dugme „Sledeća" aktivno, to jest nema u HTML-u klasu „disabled", vrši se provera broja redova i na sledećoj stranici i svim narednim stranicama ukoliko ih ima.

4) U testu za validaciju naslova kolona tabele prolazi se kroz sva polja naslovnog reda tabele i vrednosti teksta se upoređuju sa zadatim vrednostima u nizu **values**. Na stranici se najpre identifikuje tabela uz pomoć ID-a. Zatim se pravi lista **rows** svih redova u tabeli. U spoljašnjoj petlji se za svaki red izvlači kolona sa imenom taga „th", a to je, zapravo, polje iz naslovnog reda (eng. header). U unutrašnjoj petlji se poredi tekst atribut takvih polja sa vrednstima zadatim u nizu **values.**

```
@Test
public void testHeaders()
{
    String[] values = {"ID", "Prezime i ime", "JMBG", "E-mail", "Telefon"};
    WebElement table = driver.findElement(By.id("fa-dim-table"));
    List<WebElement> rows = table.findElements(By.tagName("tr"));

      for (WebElement row : rows) {
          List<WebElement> cols =
          row.findElements(By.tagName("th"));
          for (WebElement col : cols) {
          boolean match = false;
          for(int i = 0; i < values.length; i++ ){
              if (col.getText().equals(values[i]))
                      System.out.println(col.getText() + "\t" + values[i]);
                      match = true;

          }
          Assert.assertTrue(match);
          System.out.print("Nazivi kolona su korektni");
          }
      }

}
```

5) Vrednosti u tabeli su različitog formata. Za prvu kolonu očekuje se isključivo broj, za drugu se ne očekuju brojevi ili specijalni karakteri jer ih nema ni u imenima i prezimenima klijenata. Treća kolona sadrži JMBG klijenata i za taj broj postoji ograničenje da mora imati isključivo 13 cifara. Sve manje ili više ne predstavlja JMBG. Takođe se za naredne dve kolone očekuje poseban format. Za svaku e-mail adresu, podrazumeva se da sadrži bar '@' karakter, a za broj telefona da ima najmanje 9, a najviše 10 cifara. Svi ovi različiti očekivani formati se mogu opisati regularnim izrazima. Poslednja kolona treba da sadrži po dva dugmeta u svakom redu.

Naredni kôd prikazuje funkcije za verifikaciju vrednosti kolona:

```
@Test
public void testValuesFormat() {

    int rowCount = driver.findElements(
            By.xpath(".//*[@id='fa-dim-table']/tbody/tr/td[1]")).size();
    System.out.println("Broj redova je " + rowCount);

    int columnCount = driver.findElements(
            By.xpath(".//*[@id='fa-dim-table']/tbody/tr[1]/td")).size();
    System.out.println("Broj kolona je " + columnCount);
    assertEquals(6, columnCount);



    List<WebElement> firstColumnValues = driver.findElements(
            By.xpath(".//*[@id='fa-dim-table']/tbody/tr/td[1]"));
    for (WebElement value : firstColumnValues)
    {
        System.out.println(value.getText());
        assertTrue(value.getText().matches("[1-9]+"));
    }

    List<WebElement> secondColumnValues = driver.findElements(
            By.xpath(".//*[@id='fa-dim-table']/tbody/tr/td[2]"));
    for (WebElement value : secondColumnValues)
    {
        System.out.println(value.getText());
        assertTrue(value.getText().matches("[ a-zA-Z-]+"));
    }

List<WebElement> thirdColumnValues = driver.findElements(
        By.xpath(".//*[@id='fa-dim-table']/tbody/tr/td[3]"));
for (WebElement value : thirdColumnValues)
{
    System.out.println(value.getText());
    assertTrue(value.getText().matches("[0-9]{13}"));
}

List<WebElement> fourthColumnValues = driver.findElements(
        By.xpath(".//*[@id='fa-dim-table']/tbody/tr/td[4]"));
for (WebElement value : fourthColumnValues)
{
    System.out.println(value.getText());
    assertTrue(value.getText().matches("[a-zA-Z0-9\\._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}"));
}


List<WebElement> fifthColumnValues = driver.findElements(
        By.xpath(".//*[@id='fa-dim-table']/tbody/tr/td[5]"));
for (WebElement value : fifthColumnValues)
{
    System.out.println(value.getText());
    assertTrue(value.getText().matches("[0-9]{9,10}"));
}
```

```
        List<WebElement> sixthColumnValues = driver.findElements(
                By.xpath(".//*[@id='fa-dim-table']/tbody/tr/td[6]"));
        for (WebElement value : sixthColumnValues)
        {
            if(value.findElement(By.linkText("Izmeni")).isDisplayed())
            {
            System.out.println("[Izmeni] dugme je nadjeno");
            }
            else
            {
            System.out.println("[Izmeni] dugme ne postoji");
            }


            if(value.findElement(By.linkText("Detalji")).isDisplayed())
            {
            System.out.println("[Detalji] dugme je nadjeno");
            }
            else
            {
            System.out.println("[Detalji] dugme ne postoji");
            }
        }
    }
```

6) Podrazumevani redosled podataka u tabeli je opadajući po prvoj koloni. To je zato što se novi klijenti dodaju na vrh tabele, a njima se dodelje ID koji je za jedan veći od, do tada, najvećeg. Klikom na naslov kolone, vrednosti će biti uređene u rastućem poretku.

Sortiranje postoji i na ostalim kolonama. Klikom na naslov, prvo se vrednosti sortiraju u rastućem poretku, a još jednim klikom, u opadajućem.

Za sortiranje se koriste pomoćne funkcije kojima se proverava da li je poredak rastući ili opadajući. Narednom funkcijom se proverava da li je poredak elemenata rastući.

```
    public boolean chkalphabetical_order(LinkedList<String> product_names){

        String previous = ""; // empty string

        for (final String current: product_names) {
            if (current.compareTo(previous) < 0)
                return false;
            previous = current;
        }

        return true;

        }
```

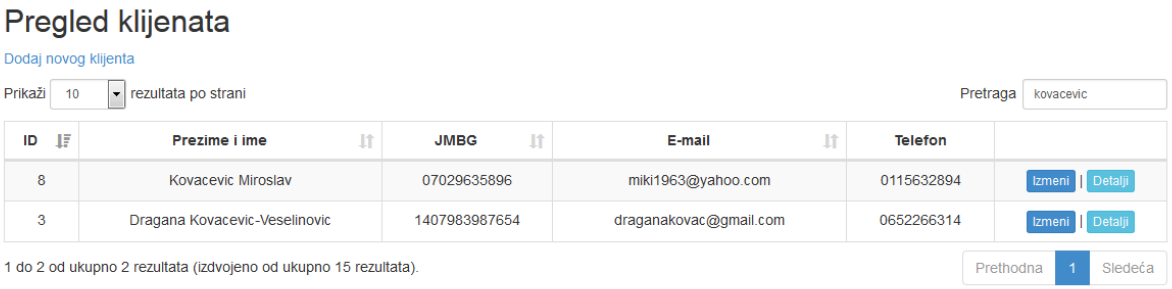
Sledeći test za svaku kolonu ispisuje vrednosti u podrazumevanom poretku. Zatim se klikom na naslov svake kolone menja redosled vrednosti i vrednosti u tom redosledu se ispisuju. U narednim koracima se validira da li je poredak vrednosti rastući.

Isti test sadrži i obrnutu proveru, a prilikom izvršavanja testa, redosled elementa u kolonama se naizmenično menja.

Provera prodrazumevanog redosleda koristi iste funkcije, ali nema početnog klika na dugme, već se poredak odmah validira.

```
@Test
public void testSort(){

    int columnCount = driver.findElements(By.xpath(".//*[@id='fa-dim-table']/tbody/tr[1]/td")).size();
    System.out.println("Broj kolona je " + columnCount);

    for (int i = 2; i < columnCount-1; i++ )
    {
    List<WebElement> ColumnValues = driver.findElements(By.xpath(".//*[@id='fa-dim-table']/tbody/tr/td["+i+"]"));
    for (WebElement value : ColumnValues)
    {
        System.out.println(value.getText());
    }

    driver.findElement(By.xpath(".//*[@id='fa-dim-table']/thead/tr/th["+i+"]")).click();

    List<WebElement> ColumnValues1 = driver.findElements(By.xpath(".//*[@id='fa-dim-table']/tbody/tr/td["+i+"]"))
    for (WebElement value : ColumnValues1)
    {
        System.out.println(value.getText());
    }

    List<WebElement> products_Webelement = new LinkedList<WebElement>();
    products_Webelement = driver.findElements(By.xpath(".//*[@id='fa-dim-table']/tbody/tr/td["+i+"]"));
    LinkedList<String> product_names =  new LinkedList<String>();
    for(int j=0;j<products_Webelement.size();j++){

        String s = products_Webelement.get(j).getText();

        product_names.add(s);

    }
```

7) Vrednosti u tabeli mogu da se pretražuju po bilo kom elementu tabele. Na slici 13 prikazan je rezultat pretrage po prezimenu klijenta.



Slika 13: Rezultat pretrage po prezimenu klijenta

Test ove funkcionalnosti proverava da, za različite vrednosti kolona koje se unose u polje „Pretraga”, sve vrednosti kolone tabele koje su prikazane, zadovoljavaju kriterijum pretrage, to jest sadrže tekst unet u polje.

Koristi se naredna pomoćna funkcija za unos teksta u polje:

```
public void enterText(String xpath_string, String text) {

    wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(xpath_string)));

    WebElement elem = driver.findElement(By.xpath(xpath_string));
    elem.clear();
    elem.sendKeys(text);
```

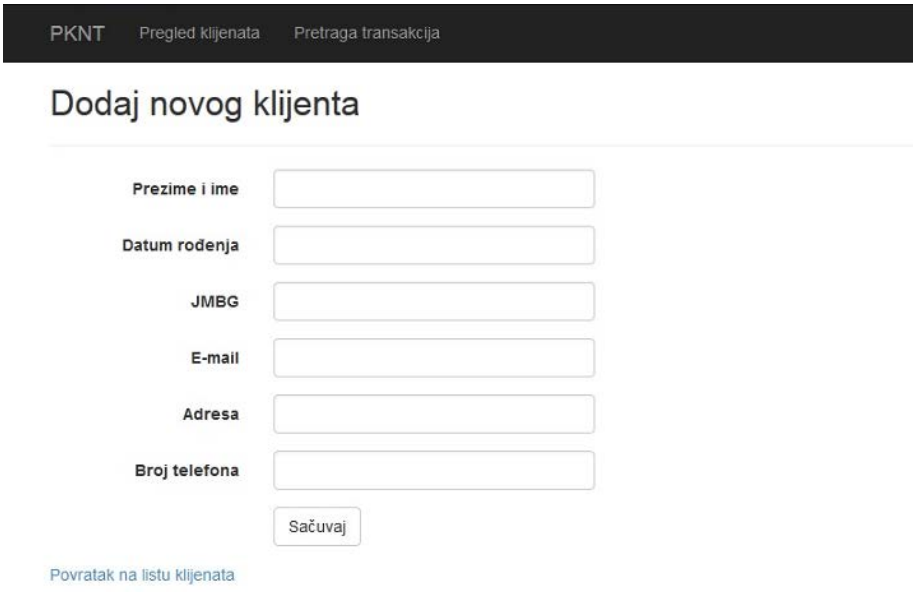
U testu se broje redovi prikazani kao rezultat pretrage (promenljiva **total_node**). Zatim se broje redovi koji sadrže tekst unet u polje za pretragu (promenljiva **total_matches**). Na kraju se upoređuju vrednosti tih promenljivih. Ukoliko su jednake, pretraga je isfiltrirala korektne redove.

U slučaju da uneti tekst nije nađen u tabeli, ispisuje se jedan red sa porukom „Nema rezultata”.

```
@Test
public void testSearch() throws Exception {

    String searchText = "063";
    enterText(".//*[@id='fa-dim-table_filter']/label/input",searchText);

    List<WebElement> nodes = driver.findElements(By.xpath(".//*[@id='fa-dim-table']/tbody/tr"));
    int total_node=nodes.size();

    int total_matches = 0;
    String rowText = "";
     for (int i=1;i<total_node+1;i++)
      {
         rowText=driver.findElement(By.xpath("//table[@id='fa-dim-table']/tbody/tr["+i+"]")).getText();
          if (rowText.matches("^[\\s\\S]*(?i:"+searchText+")[\\s\\S]*$"))
          total_matches++;
      }

     if(rowText.equals("Nema rezultata")) total_node=0;

    System.out.println(total_node);
    System.out.println(total_matches);


    assertEquals(total_node,total_matches);
}
```

Stranica „Dodaj novog klijenta”

Osnovna funkcionalnost stranice „Dodaj novog klijenta” jeste unos vrednosti u polja „Prezime i ime”, „Datum rođenja”, „JMBG”, „E-mail”, „Adresa”, „Broj telefona” i slanje podataka u bazu, klikom na dugme „Sačuvaj” (videti sliku 14). Novi klijent bi trebalo odmah da se pojavi u tabeli sa klijentima na stranici „Pregled klijenata”.



Slika 14: Stranica za unos podataka o novom klijentu

Prisustvo elemenata i funkcionalnost linka „Povratak na listu klijenata” provereni su kroz test osnovnih funkcionalnosti. Dodatne funkcionalnosti koje treba proveriti su:

1) Validacija unosa teksta u polje i kreiranja novog klijenta
2) Validacija poruka o greškama, za slučaj kada se ne popuni neko polje

1) Test unosa teksta u polja počinje nizom vrednosti koje će biti unete. Polja na stranici su identifikovana uz pomoć imena klase „text-box” i smeštena u listu elemenata. U **for** petlji se polja automatski popunjavaju elementima niza **values.** Za unos vrednosti u polje „Datum rođenja” koristi se posebna funkcija jer ovo polje ima kalendar.

```
@Test
public void testSubmitting() throws Exception{

    String[] values = {"Zoric Aleksandar", "21111975", "211197512416322", "aleks.zoric@yahoo.com",
            "Kralja Milana 122, Beograd", "0115641289"};
     List<WebElement> fields = driver.findElements(By.className("text-box"));

     for(int i = 0; i < fields.size(); i++)
     {
         System.out.println(fields.get(i).getAttribute("name"));
             if(fields.get(i).getAttribute("name").equals("FA_Birth_Date"))
             {
                System.out.println(fields.get(i).getText());
              upisiTekst("FA_Birth_Date", values[i]);
               driver.findElement(By.id("FA_JMBG")).click();
             }
             else
             fields.get(i).sendKeys(values[i]);
         }

     driver.findElement(By.id("submit-button")).click();

     Assert.assertEquals(driver.getCurrentUrl(), "http://localhost:46003/FA_Dim");
     System.out.println("Izmene su sacuvane i korisnik je vracen na stranicu sa pregledom klijenata");

     List<WebElement> columns = driver.findElements(By.xpath(".//*[@id='fa-dim-table']/tbody/tr[1]/td"));

     for (WebElement col: columns)
     {
         boolean match = false;
         for (int i = 0; i < values.length; i++)
       {
           if (col.getText().equals(values[i]))
               System.out.println(col.getText() + "\t" + values[i]);
                match = true;
       }
       Assert.assertTrue(match);
    }
    System.out.println("Vrednosti u tabeli poklapaju se sa unetim vrednostima");
}
```

Kada su sva polja popunjena i pritisne se dugme „Sačuvaj”, korisnik će automatski biti prebačen na stranicu „Pregled klijenata”. To se, takođe, validira u testu, komandom

```
Assert.assertEquals(driver.getCurrentUrl(),
"http://localhost:46003/FA_Dim");
```

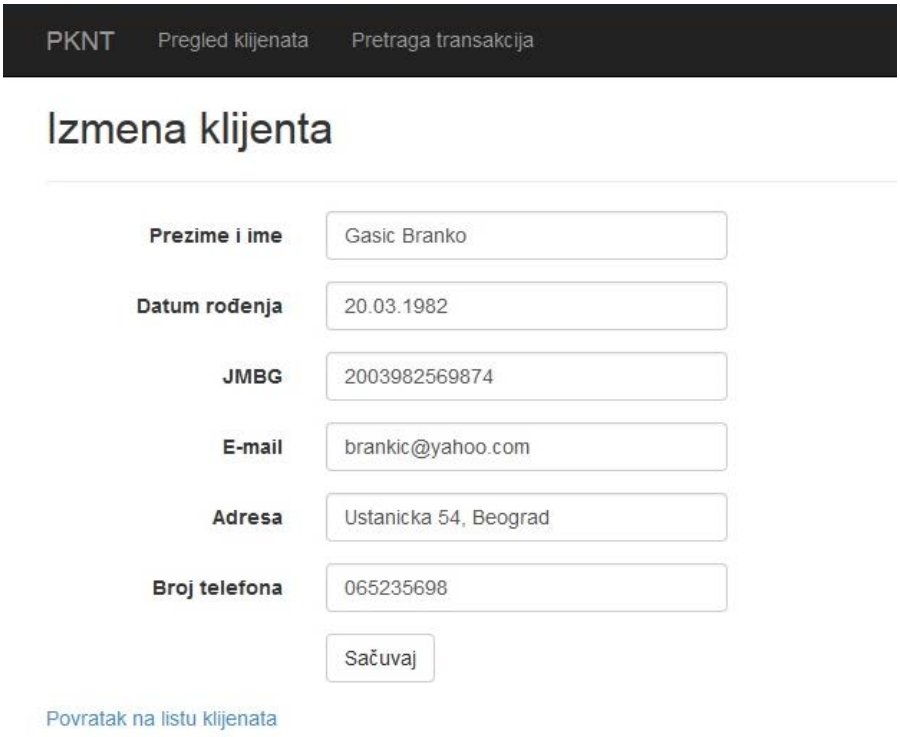
Nakon toga, sledi provera da se vrednosti kolona u prvom redu tabele na „Pregled klijenata” stranici poklapaju sa unetim vrednostima.

2) Unos vrednosti u polja na stranici „Dodaj novog klijenta” je obavezan za sva polja. Ukoliko se neko polje ne popuni, a korisnik klikne na dugme „Sačuvaj”, korisnik neće biti prebačen na stranicu „Pregled klijenata”, a ispod polja će se pojaviti poruka o grešci. Sledeći test validira poruku o grešci za polje „Prezime i ime”. Slično je i za sva ostala polja.

```
@Test
public void testErrorMessages() throws Exception
{

    // Verifikacija poruke o greski za polje "Prezime i ime"
    driver.findElement(By.id("FA_Full_Name")).sendKeys("");
    driver.findElement(By.id("submit-button")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("FA_Full_Name"))).getAttribute("data-val-required");
    String actualError = driver.findElement(By.id("FA_Full_Name")).getAttribute("data-val-required");
    String expectedError = "Polje \"Prezime i ime\" je obavezno!";
    Assert.assertEquals(actualError,expectedError);
        System.out.println("Poruka o greski za polje Prezime i ime je korektna");

}
```

Stranica „Izmena klijenta”

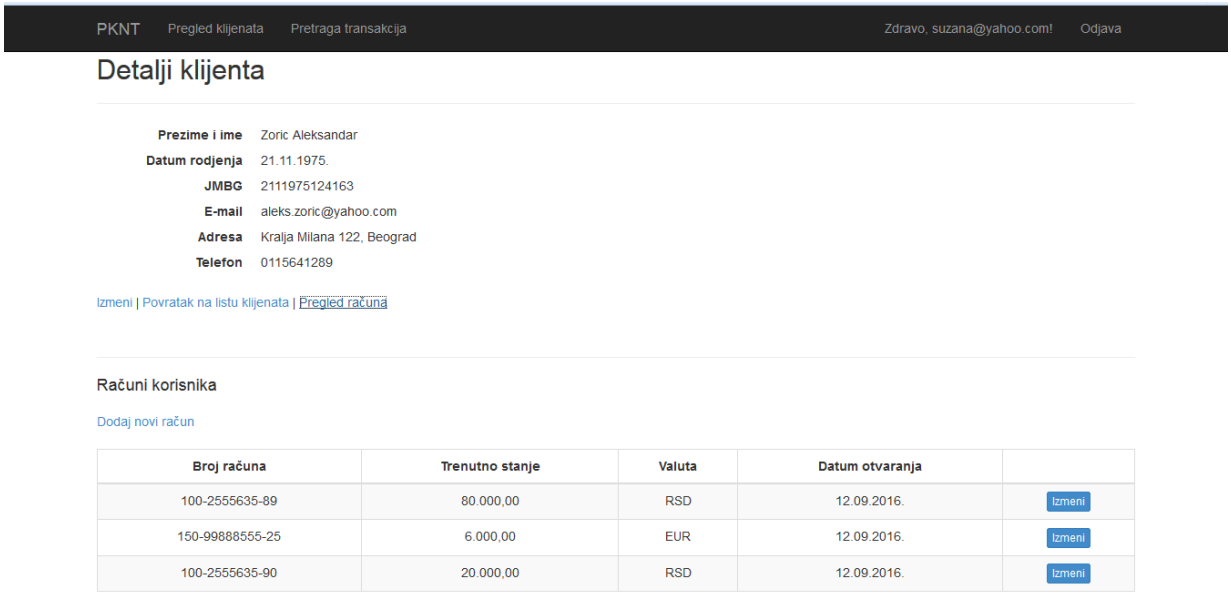
Ova stranica se otvara klikom na dugme „Izmeni” koje se nalazi u tabeli sa listom klijenata i pruža mogućnost izmene postojećih podataka o klijentima. Izgled i ponašanje elemenata na ovoj stranici su gotovo identični kao na stranici „Dodaj novog klijenta”, sa tom razlikom što su polja na početku popunjena, a ne prazna (slika 15). Svi slučajevi testiranja za ovu stranicu slični su testiranju elemenata na stranici „Dodaj novog klijenta”.



Slika 15: Stranica za izmenu podataka o klijentima

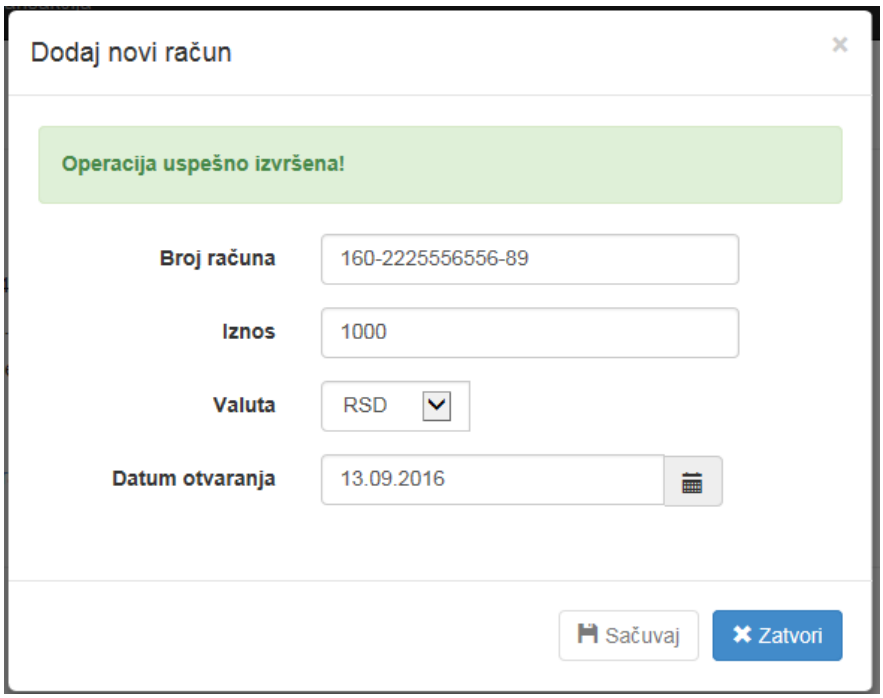
Stranica „Detalji klijenta”

Na ovoj stranici prikazani su detalji o klijentima – prezime i ime, datum rođenja, JMBG, e-mail adresa, adresa i telefon. Klikom na link „Pregled računa”, otvara se tabela sa računima klijenta, koja prikazuje brojeve računa, trenutno stanje, valutu i datum otvaranja računa (slika 16).



Slika 16: Stranica sa detaljima klijenta

Klikom na link „Dodaj novi račun”, iskače prozor sa poljima za unos broja računa i iznosa i za izbor valute i datuma otvaranja. Pritiskom na dugme „Sačuvaj”, detalji o novom računu upisuju se u bazu i pojavljuje se poruka „Operacija uspešno izvršena” (slika 17). Da bi se vratio na prethodni prozor, korisnik mora da klikne na dugme „Zatvori”.



Slika 17: Prozor za unos novog računa

Na stranici „Detalji klijenta" dostupan je i link za izmenu računa, kojim se otvara slični iskačući prozor, sa već popunjenim poljima.

Testom osnovnih funkcionalnosti izvršena je provera prisustva svih elemenata na stranici i funkcionisanje dugmadi i linkova. Detaljnije provere bi podrazumevale:

1) Validaciju broja kolona tabele
2) Validaciju naslova kolona tabele
3) Validaciju formata vrednosti u tabeli
4) Validaciju unosa vrednosti u polja na Dodaj/Izmeni račun iskačućim prozorima i kreiranja/izmene računa
5) Validaciju poruka na iskačućim prozorima

Za validaciju tačaka 1) - 3) koristi se potpuno isti postupak kao na stranici „Pregled klijenata". Tačke 4) i 5) su interesantne kao primeri kako se može pristupiti iskačućim prozorima u WebDriver-u i vršiti validacija na tim prozorima.

4) Kada Web aplikacija ima više prozora ili okvira, Selenium WebDriver dodeli alfanumerički ID svakom prozoru odmah pošto je WebDriver objekat inicijalizovan. Jedinstveni ID se zove hendler. Selenium koristi tu jedninstvenu vrednost da u aplikaciji prelazi sa jednog prozora na drugi.

U primeru koji sledi, promenljivoj **parentWindowHandler** se dodeljuje vrednost hendlera glavnog prozora, koji se dobija pozivom metode **getWindowHandle().** Takođe se

pravi i niz hendlera koji pripadaju potprozorima. Sve dok postoji potprozor potprozora, korisnik je u stanju da pristupa svakom od njih uz pomoć komande

```
driver.switchTo().window(subWindowHandler);
```

Povratak na osnovni prozor vrši se komandom

```
driver.switchTo().window(parentWindowHandler);
```

Dok je na potprozoru, korisnik može da pristupa elementima kao da se nalazi na glavnom prozoru.

```
link.click();

Thread.sleep(3000);


String parentWindowHandler = driver.getWindowHandle(); // Store your parent window
String subWindowHandler = null;

Set<String> handles = driver.getWindowHandles(); // get all window handles
Iterator<String> i = handles.iterator();
while (i.hasNext()){
    subWindowHandler = i.next();
}
driver.switchTo().window(subWindowHandler);
driver.findElement(By.xpath(".//*[@id='AccountNumber']")).sendKeys("100-2555635-90");
driver.findElement(By.xpath(".//*[@id='Amount']")).sendKeys("20000");


driver.findElement(By.xpath(".//*[@id='add-new-account-modal']/div/form/div/div[3]/button[1]")).click();
String alt = driver.findElement(By.xpath(".//*[@id='alert-success']/strong")).getText();
System.out.println("Poruka " +  alt);
driver.findElement(By.xpath(".//*[@id='add-new-account-modal']/div/form/div/div[3]/button[2]")).click();

driver.switchTo().window(parentWindowHandler);
```

Stranica „Pretraga transakcija”

Pored provere osnovnih funkcionalnosti i funkcionalnosti opisanih u prethodnim primerima, na stranici „Pretraga transakcija” treba još validirati:

1) Odnos između vrednosti polja za unos teksta
2) Vrednosti u tabeli u zavisnosti od postavljenih parametra

1) Stranica „Pretraga transakcija” sadrži dva para konteksno povezanih polja. To su polja „Početak perioda”, „Kraj perioda” i polja „Ukupan iznos veći od”, „Ukupan iznos manji od”. Kod ovih polja treba validirati da početak perioda ne može biti veći od kraja perioda, kao i da „ukupan iznos veći od” ne može biti manji od „ukupnog iznosa manji od”.

U primeru testa koji sledi, kod polja za iznos, prvo se unese bilo koja vrednost u prvo polje, a onda manja vrednost u drugo polje. Validira se da li se pojavila poruka o grešci da tako nešto nije dopušteno i da li je polje ostalo prazno. Slično je i za datume.

```
@Test
public void testFields() throws Exception {


    enterText(".//*[@id='MoneyBigger']","2000");

    assertEquals("2000",driver.findElement(By.xpath(".//*[@id='MoneyBigger']")).getAttribute("value"));

    enterText(".//*[@id='MoneyLess']","1000");

    assertEquals("",driver.findElement(By.xpath(".//*[@id='MoneyLess']")).getAttribute("value"));

}
```

2) Validacija vrednosti u tabeli, u odnosu na izabrane parametre, izvršava se koracima:

```
List<WebElement> nodes = driver.findElements(By.xpath(".//*[@id='transaction-search-table']/tbody//tr/td[3]"));
int total_node=nodes.size();


for (int i=0;i<total_node;i++)
  {
    iznosIzTabele=nodes.get(i).getText();
    iznosIzTabele = iznosIzTabele.replace(".","");
    iznosIzTabele = iznosIzTabele.replace(",",".");


    assertTrue(Float.parseFloat(iznosIzTabele)>Float.parseFloat(pocetniIznos));
    assertTrue(Float.parseFloat(iznosIzTabele)<Float.parseFloat(krajnjiIznos));
  }
```

Posle unošenja datuma i iznosa u odgovarajuća polja, elementi kolone „Ukupan iznos" se smeštaju u listu, a onda se vrednost svakog elementa poredi sa početnim i krajnjim iznosom. Treba da bude veća od početnog, a manja od krajnjeg.

5 Zaključak

Iako svaki programer teži da napravi softver bez grešaka, u praksi je to gotovo nemoguće. U ovom radu su opisani različiti načini testiranja softvera koji obezbeđuju da se kupcima isporuči proizvod koji odgovara njihovim početnim zahtevima. Takođe, ranije otkrivanje nedostataka smanjuje troškove razvoja, pa je vrlo važno da se potencijalni problemi što pre uoče i prevaziđu. Svako eventualno kašnjenje i neispunjavanje dogovorenih rokova utiče na cenu proizvoda, a ozbiljni propusti u softveru bi mogli imati i katastrofalne posledice i ugroziti ljudske živote. Ovo su neki od razloga zašto se sve više pažnje posvećuje testiranju softvera i zašto se ulažu napori da se ono učini što efikasnijim. Poslednjih par godina uočava se trend skraćivanja ciklusa razvoja softvera, tako da je moguće na svakih par nedelja isporučiti gotov proizvod klijentu. Sve to ne bi bilo moguće da se testiranje softvera ne automatizuje.

Na primeru aplikacije „PKNT" prikazan je proces automatskog testiranja korišćenjem Selenium alata i njegovih komponenti Selenium IDE i Selenium WebDriver. Napisani su i implementirani automatski testovi za testiranje ponašanja aplikacije koja se koristi za dodavanje i izmenu podataka o klijentima i njihovim računima, kao i naprednu pretragu njihovih transakcija. Aplikacija je napisana korišćenjem programskog jezika C#, a testovi jezikom Java.

Selenium omogućava pisanje testova na velikom broju programskih jezika i lako se integriše sa drugim alatima zbog čega predstavlja jedno od najznačajnijih rešenja u oblasti automatizovanog testiranja. Za različite potrebe testiranja, Selenim nudi veliki broj eksternih biblioteka, što u mnogome olakšava proces pisanja testova. Takođe, Selenium je besplatan i otvorenog koda, može se modifikovati prema potrebama korisnika, što je još jedan od razloga velike populatnosti Selenium-a širom sveta.

6 Literatura

[1] Daniel Galin, *Software Quality Assurance – From theory to implementation*, 2004.

[2] Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black – *Foundations of Software Testing - ISTQB certification*

[3] Learn software testing - http://www.tutorialspoint.com/software_testing/

[4] Pripremni material za ISTQB Certification ispit - http://istqbexamcertification.com/what-is-fundamental-test-process-in-software-testing/

[5] Burnstein, *I., Practical Software Testing: A Process-Oriented Approach.* 2003.

[6] Glenford J. Myers, *The Art of Software Testing*, 2012.

[7] Wikipedia - https://en.wikipedia.org/wiki/Selenium_%28software%29

[8] Zvanična Selenium dokumentacija - http://docs.seleniumhq.org/

[9] David Burns, *Selenium 2 Testing Tools Beginner's Guide*, 2012.

[10] Unmesh Gundecha, *Selenium Testing Tools Cookbook*, 2012.

[11] Vodič za instalaciju Selenium WebDriver -a - http://www.guru99.com/installing-selenium-webdriver.html